Analyses statistiques des communications sur puce

Antoine Scherrer

N° attribué par la bibliothèque : 06ENSL0 393

THESE

en vue d'obtenir le obtenir le grade de

Docteur de l'Ecole Normale Supérieure de Lyon

spécialité : Informatique

Laboratoire de l'Informatique du Parallélisme

école doctorale de : Mathématique et Informatique Fondamentale

présentée et soutenue publiquement le 11 décembre 2006

par Monsieur Antoine SCHERRER

Titre :

Analyses statisitiques des communications sur puce

Directeur de thèse :	Tanguy RISSET
Après avis de :	Monsieur Alain GREINER, Membre/Rapporteur Monsieur Patrick THIRAN, Membre/Rapporteur

Devant la commission d'examen formée de :

Monsieur Patrice ABRY, Invité
Monsieur Paul FEAUTRIER, Membre
Monsieur Antoine FRABOULET, Membre
Monsieur Alain GREINER, Membre/Rapporteur
Monsieur Tanguy RISSET, Membre
Monsieur Patrick THIRAN, Membre/Rapporteur

Remerciements

Je tiens à remercier en premier lieu mes directeurs de thèse pour leur constant soutient, leur aide, leur sympathie et leur compréhension. Ces qualités m'ont beaucoup aidé à tous les niveaux et dans toutes les étapes de l'accomplissement de ce travail.

Je voudrais aussi remercier chaleureusement les personnes avec lesquelles j'ai beaucoup collaboré durant ces trois années : Patrice Abry et Pierre Borgnat du laboratoire de Physique de l'ENS de Lyon, ainsi que Philippe Owezarski et Nicolas Larieu du LAAS.

Je souhaite aussi remercier Alain Greiner et Patrick Thiran, qui ont accepté de relire ce manuscrit. Leurs remarques m'ont permis de considérablement en améliorer la qualité et d'envisager des perspectives de recherche nouvelles.

Enfin, je voudrais remercier tous ceux qui m'ont soutenu tout au long de ma thèse, que soit d'une point de vue scientifique ou d'un point de vue personnel. En particulier, cette thèse est dédiée aux PICASOLS, sans qui rien n'aurait été possible.

Table des matières

Notations

x	Variable scalaire x
$\mathbf{x}$	Vecteur $\mathbf{x}$
$\mathbb{P}$	Probabilité
$\mathbb{E}$	Espérance mathématique
$\|..\|$	Valeur absolue
$\log$	Logarithme népérien
$\log_2$	Logarithme à base 2
$\stackrel{\Delta}{=}$	Égale, par définition
$\stackrel{d}{=}$	Possède la même loi que
γ	Covariance
Γ	Densité spectrale de puissance
Γ_f	Fonction gamma standard

Introduction générale

Les travaux qui ont été effectués dans le cadre de cette thèse ont porté sur deux grands thèmes : l'analyse et la génération de trafic Internet (partie une) et l'analyse et la génération de trafic sur puce (partie deux). Le sujet original de la thèse concernait l'analyse du trafic ayant lieu au sein d'une puce de microélectronique, c'est à dire des communications entres les différents composants (processeurs, mémoires, accélérateurs matériels) intégrés dans un système sur puce (SoC pour *System on Chip*) et était destiné à être réalisé en étroite collaboration avec la société ST Microelectronics, qui a co-financé ces travaux. L'objectif était d'étudier des modèles de trafic pertinents et adaptés aux communications des SoC, et nous avons, dès le départ, porté notre attention sur une modélisation basée sur des processus stochastiques, principalement car ST Microelectronics ne disposait pas de tels modèles pour générer du trafic.

Notre attention s'est alors portée sur la modélisation de la propriété de longue mémoire mise en évidence par Varatkar et Marculescu dans le trafic sur puce [151]. Nous avons profité de la proximité géographique de Patrice Abry, physicien au laboratoire de physique de l'ENS de Lyon, et expert international de la modélisation de cette propriété de longue mémoire, pour étudier en détail cet aspect avec lui. N'ayant pas encore de traces de trafic issues de systèmes sur puces, nous avons travaillé sur des traces de trafic Internet, cette étape nous permettant de prendre en main les outils ainsi que de nous familiariser avec les différentes théories mises en jeu (processus stochastiques, estimation des paramètres, analyse en ondelettes, etc.). Cette collaboration a duré tout au long des trois années de la thèse, et a donné lieu aux contributions présentées dans la première partie de ce manuscrit. Nous avons en particulier étudié la problématique de la synthèse de réalisations de processus à longue mémoire non-gaussiens, et proposé une modélisation conjointe des statistiques d'ordres un et deux du trafic Internet. Cette modélisation a ensuite été utilisée pour étudier une méthode de détection d'anomalie dans le trafic Internet, en partenariat avec le projet METROSEC.

Parallèlement à ces travaux, nous avons étudié la problématique de l'analyse et la génération de trafic dans le cadre des système sur puce. La conception de ces systèmes, incluant à l'heure actuelle plusieurs millions de transistors dans quelques millimètres carrés, est très complexe et constitue un sujet actif de recherche. En particulier, l'interconnexion, c'est à dire le composant permettant à tous les composants de la puce de communiquer, est en train de vivre un changement technologique majeure avec l'introduction de véritables *réseaux sur puce*. Disposer d'outil de génération de trafic pour l'évaluation de performance de ces microréseaux est un enjeu important pour évaluer leur faisabilité et comparer les différentes proposition. Aussi, le temps de simulation d'une puce est très important lorsque les composants sont très précisément simulés, et remplacer ces derniers par des générateurs de trafic permet de réduire le temps de simulation. Nous avons pour cela mis en place un environnement de génération de trafic sur puce dans le but de prototyper l'interconnexion. Notre objectif a toujours été de générer du trafic *proche* de celui émis par le composant que l'on souhaite émuler et c'est le cadre de cette problématique que nous avons utilisé des processus stochastique pour modéliser ce trafic. Nous avons effectuer des simulations dans l'environnement de simulation académique SOCLIB, qui nous ont permis

de valider notre approche. Ces travaux sont présentés en détail dans la deuxième partie de ce manuscrit.

Le point commun de ces deux parties réside dans l'utilisation de processus stochastiques pour modéliser des *séries temporelles*[1] ayant un comportement qui échappe à une modélisation déterministe basée sur des automates d'état finis par exemple. L'évolution de la valeur d'une action en bourse, l'évolution du débit sur un lien d'un réseau, ou encore l'évolution des temps entre deux transactions processeur/mémoire consécutives sont des exemples de telles séries. L'intérêt principal de la modélisation par processus stochastique est d'extraire de la série une information pertinente et de l'utiliser en lieu et place de celle-ci. On pourra ainsi, pour représenter une série de plusieurs milliards d'échantillons, utiliser simplement un modèle paramétrique. Ce modèle pourra servir à faire de la prévision (prévoir le futur étant donné le présent), à produire des séries synthétiques (artificielles) ayant des caractéristiques statistiques proches de la série originale, ou encore à effectuer des évaluations de performance analytiques ou par simulations numériques.

Il est à noter que beaucoup de données étant le résultat d'un procédé tout à fait déterministe, comme les accès à la mémoire d'un processeur exécutant un code donné par exemple, sont de bons candidats pour une modélisation à base de processus stochastiques. On parle alors de *chaos déterministe*, c'est à dire un aléatoire apparent pourtant issu d'un procédé déterministe. Les données ne sont alors pas, à strictement parler, aléatoires, néanmoins une modélisation à l'aide de processus stochastiques peut être utile dans différents cas : lorsque le procédé déterministe de création des données est trop complexe, lorsqu'il n'est pas accessible ou encore lorsque l'on veut s'affranchir de ce procédé, c'est à dire chercher à caractériser les données sans savoir ce qui les a produites.

Il existe un grand nombre de modèles de processus stochastiques, du plus simple au plus complexe. Le choix de ce modèle dépend de l'utilisation faite de la modélisation et de la nature des données. Modéliser un trafic avec des processus stochastiques implique les étapes suivantes :

1. **Recueillir des données**. Il faut donc avoir accès à des traces de trafic, et convertir ce trafic en séries temporelles.
2. **Choisir un modèle**. A partir d'une analyse des données ainsi que d'informations sur le trafic, un modèle est sélectionné.
3. **Estimer les paramètres du modèle**. Il faut pour cela faire passer les données dans un outil d'analyse statistique permettant d'estimer les meilleurs paramètres du modèle que l'on s'est fixé, étant donnée la série temporelle.
4. **Utiliser le modèle**. Une fois que l'on dispose du modèle et de ses paramètres, on peut l'utiliser pour générer des réalisations synthétiques de celui-ci, ou effectuer des prévisions. Cette tâche n'est pas toujours aisée.

Il faut enfin, afin de valider une modélisation, disposer d'indicateurs permettant de caractériser l'adéquation du modèle aux données. Cela permet de pouvoir comparer différentes modélisations afin de sélectionner la meilleure, mais aussi d'avoir une certaine confiance en la modélisation que l'on fait. Par exemple, si on utilise des modèles de processus stochastiques stationnaires pour modéliser des données qui présentent d'importantes non-stationnarités, alors la modélisation ne sera pas valide et pourra mener à des interprétations erronées.

[1] Ces séries doivent leur nom au fait qu'elles représentent souvent l'évolution d'une grandeur au cours du temps.

Première partie

Analyse et synthèse de trafic Internet

Introduction

L'ensemble des travaux présentés dans cette partie ont été menés en étroite collaboration avec l'équipe SISYPHE du laboratoire de physique de l'ENS de Lyon.

Le trafic circulant sur les réseaux d'ordinateurs (comme le trafic Internet par exemple), est un élément déterminant de la procédure de dimensionnement de ces réseaux. De nombreux paramètres doivent être explorés (topologie, débit des liens, protocole de routage, taille des files d'attente dans les routeurs), et cette étape ayant lieu avant la mise en place du réseau, on est obligé d'utiliser des charges de trafic artificielles pour faire cette exploration et prendre les bonnes décisions. Le moyen le plus utilisé pour produire ces traces de trafic *synthétiques* est d'utiliser des réalisations de processus stochastiques. Ces processus disposent en effet de nombreux avantages, on peut effectuer dans certains cas des calculs analytiques d'estimation de performance, la variabilité du trafic peut être reproduite et surtout il est possible, à partir d'un trafic enregistré sur un réseau, de *modéliser* le trafic. Ce modèle est ensuite utilisé dans les simulations pour le dimensionnement du réseau, et bien entendu plus le modèle est adapté (les caractéristiques statistiques du trafic sont bien capturées par le modèle), plus l'évaluation de performance et le dimensionnement effectué sera pertinente. C'est pourquoi la modélisation de trafic Internet est un sujet très actif de recherche.

Cette modélisation a subi, il y a une dizaine d'année, un changement important avec l'identification d'une caractéristique statistique nouvelle dans le trafic : *l'invariance d'échelle*, qui est aussi appelée *longue mémoire* ou *autosimilarité*. Cette caractéristique est maintenant communément admise et, étant donné qu'elle a un impact fort sur les performances des réseaux (cela est l'objet d'un livre de Kihong Park [118]), il est nécessaire d'en tenir compte afin de faire une modélisation pertinente. Une autre caractéristique importante du trafic Internet est le fait qu'il soit non-gaussien, c'est à dire que la distribution du débit ne suit pas une loi Normale. De nombreux travaux ont étudié ces deux propriétés indépendamment, et notre objectif a été de proposer une modélisation *conjointe* de ces deux propriétés importantes. Nous avons aussi, afin de répondre à la problématique principale de la thèse à savoir la génération de trafic, étudié la génération de réalisation de processus non-gaussien et à longue mémoire. Enfin, la définition de cette modélisation nous a permis d'imaginer un dispositif de détection d'anomalie.

Cette partie est organisée comme suit : le chapitre 1 présente des rappels mathématiques ainsi que quelques processus stochastiques, le chapitre 2 introduit ensuite en détail la famille des processus ayant une caractéristique de longue mémoire. Ma contribution à la synthèse de processus à longue mémoire non-gaussiens est détaillée dans le chapitre 3, et des applications de ce travail à la modélisation de trafic Internet et à la détection d'anomalies sont présentées dans le chapitre 4.

Chapitre 1
Processus stochastiques

Ce chapitre présente des rappels mathématiques sur les processus stochastiques et introduit les modèles qui seront utilisés dans les sections suivantes. Toute la théorie des probabilités n'est pas reprise ici, le lecteur est renvoyé à l'excellent livre de Geoffrey R. Grimmett et David R. Stirzaker intitulé "*Probability and Random Processes*" [55] pour trouver des compléments d'informations. Les notions de bases concernant les variables aléatoires et la théorie des probabilités sont exposées dans la section 1.1, puis les processus stochastiques sont définis et caractérisés dans la section 1.2. La section 1.3 présente ensuite quelques modèles de processus stochastiques classiques et enfin la section 1.4 décrit les différentes étapes impliquées dans la modélisation d'un ensemble d'observations et la génération de réalisations synthétiques statistiquement proche de cet ensemble.

1.1 Variables aléatoires

Cette section contient des rappels sur la théorie des probabilités et en particulier sur les variables aléatoires, qui sont l'ingrédient de base des processus stochastiques.

1.1.1 Définition

Une variable aléatoire (*VA*) est une application d'un espace probabilisé (Ω, A, P) dans $\mathbb{R}$, où Ω est l'univers (ensemble des possibles), A est l'ensemble des sous-ensembles de Ω (ensemble des événements), et $\mathbb{P}$ est une application qui associe à tout élément de A une probabilité, c'est à dire un réel compris entre 0 et 1. A chaque événement noté $\omega \in A$ correspond donc une probabilité d'apparition $\mathbb{P}(\omega)$.

Une *VA* X associe à chaque événement $\omega \in A$, un nombre x appartenant à $\mathbb{R}$, $X(\omega) = x$. Comme deux événements peuvent être associés au même réel x, une *VA* est souvent définie par sa fonction inverse :

$$\forall x \in \mathbb{R}, \quad X^{-1}(x) = \omega \in A,$$

Si X prend ses valeurs dans $\mathbb{R}$, on parle de *VA* continue, et si X ne prend que ses valeurs dans $\mathbb{N}$ ou $\mathbb{Z}$, alors on parle de *VA* discrète.

1.1.2 Loi d'une variable aléatoire

La *loi* d'une *VA* est définie par deux fonctions équivalentes, la fonction de répartition (CDF pour *Cummulative Distribution Function*, notée F_X) et la fonction de masse (PDF pour *Probability Density Function*, notée f_X). La PDF caractérise la répartition des valeurs prises par une variable aléatoire X, c'est à dire pour tout x, la probabilité que X soit égal à x. La CDF est une

autre représentation de cette information, c'est à dire pour tout x, la probabilité que X soit égal à x. L'*espérance mathématique* d'un *VA* X (notée $\mathbb{E}(X)$) est quand à elle la valeur moyenne prise par la *VA*, c'est à dire la somme des valeurs que peut prendre X pondérées par leurs probabilités d'apparition. Pour qu'une variable aléatoire soit correctement définie, il faut s'assurer que la probabilité qu'une valeur apparaisse soit égale à 1 (événement certain). Cette règle est appelée *règle des probabilités totales*.

Dans le cas d'une *VA* discrète, on a :

- CDF : $\forall x \in \mathbb{N}, \quad F_X(x) = \mathbb{P}(X \leq x)$
- PDF : $\forall x \in \mathbb{N}, \quad f_X(x) = \mathbb{P}(X = x)$
- Relation entre la PDF et la CDF : $\sum_{n=-\infty}^{x} f_X(n) = F_X(x)$
- Probabilités totales : $\sum_{n\in\mathbb{Z}} f_X(n) = 1$
- Espérance : $\mathbb{E}(X) = \sum_{n\in\mathbb{Z}} n\mathbb{P}(X = n) = \sum_{n=-\infty}^{\infty} n f_X(n)$

De manière équivalente, dans le cas d'une *VA* continue, on a :

- CDF : $F_X(x) = \mathbb{P}\big(X \leq x\big), \quad x \in \mathbb{R}$
- PDF : $f_X(x) = \lim_{dx\to 0} \dfrac{F_X(x) - F_X(x+dx)}{dx} = F'_X(x)$
- Relation entre la PDF et la CDF : $\int_{-\infty}^{x} f_X(t)\,dt = F_X(x)$
- Probabilité totale : $\int_{\mathbb{R}} f_X(t)\,dt = 1$
- Espérance : $\mathbb{E}(X) = \int_{\mathbb{R}} t f_X(t)\,dt$

L'espérance est linéaire :

$$\forall X, Y, \quad \mathbb{E}(X+Y) = \mathbb{E}(X) + \mathbb{E}(Y)$$

$$\forall a, \quad \mathbb{E}(aX) = a\mathbb{E}(X)$$

Un résultat important de la théorie des probabilité est la loi des grands nombres, qui relie un ensemble de N observations d'une *VA* X notées $X(n)$, $n \in 1,..,N$ avec l'espérance de X :

$$\lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N} X(n) = \mathbb{E}\big(X\big) \tag{1.1}$$

La loi d'une *VA* peut être estimée dans des données par la fréquence d'apparition des valeurs. On peut en effet montrer à partir de la loi des grands nombres (équation 1.1), que pour un nombre suffisamment grand d'observations, la fréquence d'apparition d'une valeur converge vers sa probabilité d'apparition.

Il existe de nombreuses lois de variables aléatoires *classiques*, discrètes et continues. Une partie d'entres elles seront présentées dans le tableau 2.1 de la section 2.10 (page 43). Parmi ces lois, la loi Normale (ou Gaussienne) jouit d'une position particulière puisque le théorème de la limite centrale stipule que la variable aléatoire S_N, somme de N *VA* indépendantes et identiquement distribuées notées $\{X[i]\}_{i=1,...,N}$, $S_N = \sum_{i=1}^{N} X[i]$ tend vers une loi Normale de moyenne μ et de

variance σ^2 lorsque N tend vers l'infini. μ et σ^2 sont respectivement l'espérance et la variance des $X[i]$.

On parle de la *queue* de la loi d'une *VA* X pour dénoter la décroissance de sa PDF $f_X(x)$ lorsque x tend vers l'infini. On parle d'une queue *lourde* dans le cas où la décroissance de la PDF est équivalente à une loi de puissance :

$$f_X(x) \underset{x\to\infty}{\sim} a|x|^{-\alpha}, \quad 0 < \alpha \leq 1$$

Cette propriété caractérise le fait que la *VA* pourra prendre, avec une probabilité non négligeable, des valeurs importantes, et c'est donc une caractéristique importante. En particulier, la distribution des tailles de fichiers qui circulent sur le réseau Internet possède cette propriété [78, 118, 41], et cela possède un impact important sur les performances du réseau. La loi de Pareto (voir section 2.10) est un exemple de loi à queue lourde.

1.1.3 Moments d'une variable aléatoire

Le *moment d'ordre k* d'une variable aléatoire est une quantité caractérisant l'espérance de la variable aléatoire X élevée à la puissance k :

$$\mathbb{E}\left(X^k\right) = \int_{-\infty}^{\infty} x^k f_X(x)\,dx \text{ ou } \sum_{l=-\infty}^{\infty} l^k f_X(l) \text{ si la VA est discrète}$$

Le *moment contré d'ordre k* de la *VA* X est le moment d'ordre k de la *VA* $(X - \mathbb{E}(X))$. On peut citer par exemple :

- la moyenne (Moment d'ordre 1) : $\mathbb{E}(X)$
- la variance (Moment centré d'ordre 2) : $\mathrm{Var}(X) = \mathbb{E}\left((X - \mathbb{E}(X))^2\right) = \mathbb{E}(X^2) - \mathbb{E}(X)^2$

1.1.4 Couple de variables aléatoires

Soit deux *VA* X et Y, on peut définir la *loi conjointe* du couple (X, Y) :

- CDF : $F_{XY}(x, y) = \mathbb{P}(X \leq x, Y \leq y), \quad (x, y) \in \mathbb{R}^2$
- PDF : $f_{XY}(x, y) = F_{XY}(x, y)'$

Deux *VA* X et Y sont dites indépendantes, si on a :

$$E(XY) = \mathbb{E}(X)\mathbb{E}(Y)$$

Le *coefficient de covariance* de deux *VA* X et Y est un réel défini comme suit :

$$\mathrm{Cov}(X, Y) = \mathbb{E}\left((X - \mathbb{E}(X))(Y - \mathbb{E}(Y))\right) = \mathbb{E}(XY) - \mathbb{E}(X)\mathbb{E}(Y)$$

La valeur absolue de $\mathrm{Cov}(X, Y)$ sera d'autant plus élevée que les variables seront *corrélées* (c'est à dire que l'influence réciproque de la valeur de X sur la valeur de Y sera importante). En particulier, si les *VA* sont indépendantes, les *VA* ne s'influencent pas et on a : $\mathrm{Cov}(X, Y) = 0$. Le coefficient de covariance peut être positif (X et Y s'influencent alors dans le même sens, c'est à dire qu'une valeur élevée de X va avoir tendance à forcer Y à avoir aussi une valeur élevée et réciproquement) ou négatif (une valeur élevée de X va avoir tendance à forcer Y à avoir une valeur petite et réciproquement).

On définit aussi le coefficient de corrélation ρ_{XY} de deux *VA* X et Y qui est une version normalisée du coefficient de covariance (il est compris entre -1 et 1) :

$$\rho_Y = \frac{\mathbb{E}\big((X-\mathbb{E}X)(Y-\mathbb{E}Y)\big)}{\sqrt{\mathrm{Var}(X)}\sqrt{\mathrm{Var}(Y)}} = \frac{\mathrm{Cov}(X,Y)}{\sqrt{\mathrm{Var}(X)}\sqrt{\mathrm{Var}(Y)}}$$

1.2 Processus stochastiques

Les processus stochastiques sont un outil de modélisation, et ils sont utilisés dans un grand nombre de domaines scientifiques différents. Cette section les définit et en décrit les principales caractéristiques.

1.2.1 Définition

Un processus stochastique est une suite de variables aléatoires $\{X(t)\}_{t\in\mathbb{N}\,\mathrm{ou}\,\mathbb{R}}$ [55]. Si $t \in \mathbb{R}$, le processus est dit à temps continu, et si $t \in \mathbb{N}$ il est dit à temps discret et on le notera alors $\{X[n]\}_{n\in\mathbb{N}}$. Il est à noter que le mot temps utilisé ici ne désigne pas forcément un temps physique. Si les *VA* constituant le processus sont discrètes alors on parle de processus à valeurs discrètes, si elles sont continues on parle de processus à valeurs continues. De par la nature des données que nous avons utilisées, que ce soit du trafic sur puce (séquence du nombre de cycles d'horloge entre deux communications) ou du trafic Internet (séquence du nombre d'octets transférés dans des fenêtres temporelles), nous limiterons notre étude aux processus à temps discrets (à valeurs discrètes ou continues).

Il est important de faire la distinction entre le processus stochastique en lui-même qui est un objet mathématique dont on peut définir les caractéristiques et dérivés des propriétés, et la *réalisation* d'un processus stochastique, qui est une suite de nombres aléatoires possédant des propriétés directement liées à celles de l'objet mathématique. Cette suite de nombres est par exemple une fonction de $\mathbb{N}$ dans $\mathbb{R}$, si le processus est à temps discret et à valeurs continues.

Remarque : Nous utiliserons dans ce texte indifféremment la notation X ou $\{X[n]\}_{n\in\mathbb{N}}$ pour se référer au processus stochastique X. Nous réserverons aussi l'utilisation des crochets uniquement pour parler de variable aléatoire ($X[n]$ par exemple). Une réalisation du processus stochastique X sera notée avec des parenthèses $X(n)$.

1.2.2 Statistiques d'ordre 1

La loi d'un processus stochastique $\{X[n]\}_{n\in\mathbb{N}}$ est, par définition et par extension de l'étude d'un couple de variables aléatoires (voir section 1.1.4), la loi conjointe de l'ensemble des variables aléatoires $X[n]$ du processus $\{X[n]\}_{n\in\mathbb{N}}$ [55]. C'est donc une fonction d'un vecteur $\mathbf{x} = (x_1, x_2, \ldots)$:

- CDF : $F_X(\mathbf{x}) = \mathbb{P}\big(X[1] \le x_1, X[2] \le x_2, \ldots\big)$

Cette définition stricte est trop complexe pour être utilisée en pratique, c'est pourquoi on considérera des fonctions de répartition et de densité d'une seule variable, faisant ainsi l'hypothèse que toutes les variables constituant le processus suivent la même loi. On parlera alors de loi marginale du processus :

- CDF : $\forall (k_1,k_2) \in \mathbb{N}, \forall x \in \mathbb{R}, \quad F_{X[k_1]}(x) = \mathbb{P}\big(X[k_1] \le x\big) = F_{X[k_2]}(x) \stackrel{\Delta}{=} F_X(x)$
- PDF : $\forall x \in \mathbb{R}, \quad f_X(x) = F'_X(x)$

On peut, par analogie avec une variable aléatoire, approximer la loi marginale d'un processus par la fréquence d'apparition de chaque valeur dans un ensemble d'observations. Le processus pouvant potentiellement prendre un grand nombre de valeurs, on peut, afin d'avoir une représentation plus compacte de la PDF, définir un ensemble d'intervalles, et calculer la fréquence d'apparition sur ces intervalles plutôt que sur les valeurs directement. On obtient ainsi *histogramme empirique*, qui est une image de la loi marginale d'un ensemble d'observations. Une illustration d'un histogramme empirique sur la figure 1.6 (voir section 1.4.2, page 26).

On peut aussi, à partir de l'approximation ainsi faite de la fonction de densité, calculer les *moments* du processus $\{X[n]\}_{n\in\mathbb{N}}$ de la même manière que pour une variable aléatoire.

Remarque : Le calcul de l'approximation de la PDF ainsi que des moments n'est valable que si le processus que l'on étudie est *stationnaire* (voir section 1.2.4).

La PDF observée dans un ensemble d'observations peut ensuite soit être gardée comme elle est (les paramètres du modèle sont alors l'ensemble des valeurs de l'histogramme empirique), soit être modélisée par une loi marginale classique [44] telle que la loi Normale, la loi exponentielle, etc.(le tableau 2.1 donne une liste de quelques lois marginales classiques). On aura dans ce cas une expression mathématique de la PDF [44], et on peut, par une procédure de *fit* statistique (voir section 1.4.2), trouver les meilleurs paramètres de la loi pour l'ensemble d'observations.

1.2.3 Statistiques d'ordre 2 (covariance)

La structure d'auto-corrélation d'un processus est une caractéristique importante. Elle est le reflet de la *mémoire* du processus, c'est à dire de comment les valeurs sont corrélées (la valeur de l'une influence la valeur de l'autre), en fonction de leur écart. Cette caractéristique ne nous donne aucune information sur les valeurs prisent par le processus, mais sur leur *influence mutuelle* au sein de celui-ci. Dans le cadre de l'étude de trafic, cette caractéristique permet, entre autres, de décrire un comportement de type *rafale* ($burst$ en anglais).

La fonction d'auto-corrélation du processus X notée ρ_X est une fonction de deux variables $(n,m)\in\mathbb{N}^2$ caractérisant le coefficient de corrélation des *VA* $X(n)$ et $X(m)$:

$$\rho_X(n,m)=\frac{\mathrm{Cov}(X[n],X[m])}{\sqrt{\mathrm{Var}(X[n])}\sqrt{\mathrm{Var}(X[m])}}=\frac{\mathbb{E}\big((X[n]-\mathbb{E}X[n])(X[m]-\mathbb{E}X[m])\big)}{\sqrt{\mathrm{Var}(X[n])}\sqrt{\mathrm{Var}(X[m])}}$$

On peut de même définir la fonction d'auto-covariance :

$$\gamma_X(n,m)=\mathrm{Cov}(X[n],X[m])=\mathbb{E}\Big(\big(X[n]-\mathbb{E}(X[n])\big)\big(X[m]-\mathbb{E}(X[m])\big)\Big) \tag{1.2}$$

On a, dans le cas où la variance du processus X est constante et égale à $\mathrm{Var}(X)$:

$$\rho_X(n,m)=\frac{\gamma_X(n,m)}{\mathrm{Var}(X)}$$

Nous parlerons indifféremment dans ce texte de *structure de corrélation* et de *covariance* pour se référer à la fonction d'auto-covariance du processus, qui sera notée γ. Une propriété importante de cette fonction est que si les variables aléatoires du processus sont indépendantes, alors elle est nulle partout sauf au point $(0,0)$, où elle vaut la variance.

La covariance d'un processus est en général décroissante. Le contraire signifierait que les variables aléatoires du processus seraient d'autant plus corrélées qu'elles sont éloignées, ce qui

n'est pas envisageable pour les données que nous allons analyser. En effet, à partir d'un certain écart de temps $|n-m|$, on s'attend à ce que les variables aléatoires ne soient plus corrélées, ce qui revient à dire, dans le cadre de l'analyse du trafic, que le trafic à l'instant n est complètement indépendant du trafic à l'instant $n+m$, pour m assez grand.

Tout comme la loi marginale, on peut décider de *modéliser* la covariance observée dans des données. Il existe en effet différents modèles de covariance, dont certains seront détaillés plus loin (voir section 2.10).

Remarque : Les modélisations des statistiques d'ordre un (loi marginale) et d'ordre deux (covariance) d'un processus peuvent être effectuées de manière indépendante. Il faut néanmoins prendre garde car certaines formes de covariances ont une influence sur la performance des estimateurs des statistiques d'ordre 1, et vice et versa.

1.2.4 Stationnarité

La stationnarité d'un processus en est une propriété fondamentale. Elle indique si les caractéristiques de celui-ci changent avec le temps ou non. Si le processus n'est pas stationnaire alors les informations recueillies dans les données peuvent être mal interprétées. Par exemple, si on observe dans les données deux *phases* avec des moyennes clairement distinctes, alors les informations telles que la loi marginale ou la structure de corrélation, calculées sur l'ensemble des données n'ont plus de sens. Il faudra dans ce cas séparer les phases et utiliser un modèle différent pour chacune d'entre elles. Ce point sera discuté dans le cadre du trafic sur puce dans la section 6.2.3, page 128.

Formellement, il existe deux types de stationnarité pour un processus stochastique $\{X[n]\}_{n\in\mathbb{N}}$.

- **Stationnarité au sens strict** : La loi marginale de tout vecteur aléatoire de taille N quelconque est invariante :

$$\forall N \in \mathbb{N}, \quad \forall (n,m) \in \mathbb{N}^2,$$

$$\mathbb{P}\big(X[n] \leq x_1, \ldots, X[n+N] \leq x_N\big) = \mathbb{P}\big(X[n+m] \leq x_1, \ldots, X[n+m+N] \leq x_N\big)$$

  En pratique cette définition s'avère trop restrictive, c'est pourquoi on se limitera à la stationnarité au sans large.

- **Stationnarité au sens large** : Toutes les variables aléatoires ont même moyenne et la covariance (voir section 1.2.3) ne dépend que de *l'écart de temps* entre les variables aléatoires $X[n]$ et $X[m]$, et non de l'origine. C'est à dire qu'on a : $\gamma_X(n,m) = \gamma_X(0,|n-m|) \overset{\Delta}{=} \gamma_X(|n-m|)$. Cela signifie que la structure de corrélation des variables aléatoires ne dépend pas du temps auquel on se place. La covariance est alors une fonction d'une seule variable $k = |n-m|$ et sera notée $\gamma_X(k)$ par la suite.

Nous ne considérerons que des processus stationnaires au sens large dans les travaux présentés ici. Il est à noter que le théorème de l'ergodicité [55] étend la validité de la loi de grands nombres aux processus stationnaires au sens large, c'est à dire que la correspondance entre la fréquence d'apparition des valeurs et la loi marginale reste valide. En pratique, la covariance d'un processus stationnaire au sens large ralentit la converge de la loi des grands nombres, c'est pourquoi il faut plus d'échantillons pour avoir une estimation correcte (voir section 1.4.2).

1.2.5 Spectre et densité spectrale de puissance

La transformée de Fourier d'un signal à temps discret (échantillonné) $s(n), n \in \mathbb{N}$ est une représentation du contenu fréquentiel de ce signal. Cette représentation est intéressante car elle permet d'avoir un autre point de vue sur le signal. En particulier, certaines caractéristiques de celui-ci sont plus facilement identifiables avec cette représentation. Elle est définie, pour chaque fréquence ν, comme la convolution du signal avec l'exponentielle complexe de fréquence ν :

$$\forall \nu \in \mathbb{N}, \quad TF(\nu) = \sum_{n \in \mathbb{N}} s(n) e^{-2i\pi\nu n} \tag{1.3}$$

Cette transformation est réversible (toute l'information du signal est contenue dans sa transformée de Fourier), et la transformation inverse est définie de manière symétrique :

$$\forall t \in \mathbb{N}, \quad s(n) = \frac{1}{2\pi} \sum_{\nu \in \mathbb{N}} TF(\nu) e^{-2i\pi\nu n}$$

Le spectre $S(\nu)$ (ν représente toujours la fréquence) d'un signal $s(n)$, $n \in \mathbb{N}$ est le module de sa transformée de Fourier ($S(\nu) = |TF(\nu)|$ et la *densité spectrale de puissance* (DSP), notée $\Gamma(\nu)$ dans ce texte, est le carré de ce spectre ($\Gamma(\nu) = |TF(\nu)|^2 = S(\nu)^2$) et représente comment la puissance du signal est répartie en fréquence (contribution de chaque bande de fréquence à la puissance totale). Dans le cas d'un signal aléatoire, comme la réalisation d'un processus stochastique par exemple, on peut montrer (Théorème de Wiener-Khinchine [55]) que la densité spectrale de puissance notée ici $\Gamma(\nu)$ (ν représente les fréquences) est équivalente à la transformée de Fourier de la covariance $\gamma(k)$ du processus :

$$\Gamma(\nu) = \frac{1}{2\pi} \sum_{k \in \mathbb{N}} \gamma(k) e^{-2i\pi\nu k} \tag{1.4}$$

La densité spectrale de puissance est donc une représentation de la covariance dans le domaine fréquentiel, et la connaissance de l'une entraîne la connaissance de l'autre. Cette fonction est intéressante car il est souvent utile de caractériser les propriétés d'une fonction (ici la fonction de covariance) en analysant son contenu fréquentiel. Dans le cas des processus ARMA (voir section 1.3.2) et FARIMA (voir section 2.6) par exemple, la covariance est définie par une forme mathématique de la densité spectrale de puissance, et il n'existe pas de formule analytique de la covariance.

1.2.6 Espaces fonctionnels et loi d'échelle

Afin d'obtenir différentes représentations d'une réalisation notée $X_r(n)$ d'un processus stochastique X, il est intéressant de considérer X_r comme une fonction du temps, et d'utiliser alors la théorie des espaces fonctionnels pour l'exprimer dans différentes bases. Ceci permet d'avoir des représentations différentes du processus et peut aider à faire ressortir certaines caractéristiques comme la longue mémoire (voir chapitre 2). La transformée de Fourier introduit dans la section précédente est un exemple d'une telle représentation, la base utilisée étant la famille des exponentielles complexes. Ce genre de transformation va aussi nous permettre de définir le *processus agrégé d'ordre m*, qui sera beaucoup utilisé pour caractériser le trafic tout au long de ce texte.

Considérons donc l'espace fonctionnel des fonctions d'énergie finie définies sur $\mathbb{N}$: $L^2(\mathbb{N})$. Une fonction f de cet espace doit satisfaire la condition suivante (énergie finie) :

$$\sum_{n \in \mathbb{N}} f^2(n) < \infty$$

Munissons cet espace d'un produit scalaire (noté $< .,. >$) :

$$\forall f, g \in L^2(\mathbb{N}) < f, g >= \sum_{n\in\mathbb{N}} f(n)g(n)$$

On peut alors, en utilisant la théorie des espaces fonctionnels, exprimer toute fonction $f \in L^2(\mathbb{N})$ dans différentes bases. Une base de $L^2(\mathbb{N})$ est une famille libre et génératrice de cet espace, comme dans le cas d'un espace vectoriel. Les éléments de cette famille sont néanmoins des fonctions, et si $\{g_k(n)\}_{k\in\mathbb{N}}$ est une base orthogonale de $L^2(\mathbb{N})$, alors on a la décomposition suivante :

$$f(n) = \sum_{k\in\mathbb{N}} < f, g_k > g_k(n) \tag{1.5}$$

Une base orthogonale classique de cet espace est la base de Dirac :

$$\{\delta_k(n)\}_{k\in\mathbb{N}}, \quad \delta_k(n) = \begin{cases} 1 & \text{si } n = k \\ 0 & \text{sinon} \end{cases} \tag{1.6}$$

Si on Considère une réalisation X_r d'un processus stochastique comme une fonction de $L^2(\mathbb{N})$, $X_r(n)$ est alors le $n^{\text{ième}}$coefficient de cette réalisation exprimée dans la base de Dirac de $L^2(\mathbb{N})$. On peut vouloir, afin d'obtenir une autre représentation de la fonction X_r, l'exprimer dans une autre base de $L^2(\mathbb{N})$.

Par exemple, la transformée de Fourier correspond à la projection de la fonction X_r sur l'espace engendré par la famille des exponentielles complexes ($\{e^{-2i\pi n}\}_{n\in\mathbb{N}}$), qui est une base de $L^2(\mathbb{N})$. On a bien comme dans l'équation (1.3) :

$$TF(\nu) =< X_r, e^{-2i\pi n} >= \sum_{n\in\mathbb{N}} X_r(n)e^{-2i\pi\nu n}$$

Un autre exemple classique de changement de base est utilisé pour définir le processus agrégé d'ordre m noté $X^{(m)}$. Considérons pour cela la famille des fonctions *boîtes* (voir figure 1.1) définie ainsi :

$$\{g_{k,m}(n)\}_{k\in\mathbb{N}}, \quad g_{k,m}(n) = \begin{cases} 1/\sqrt{m} & \text{si } k \le n < k+m \\ 0 & \text{sinon} \end{cases} \tag{1.7}$$

Cette famille forme une base orthogonale de $L^2(\mathbb{N})$ appelée base de Haar et on a donc l'expression de X_r dans cette base notée $X^{(m)}$:

$$X^{(m)}(n) = \sum_{k\in\mathbb{N}} < X_r, g_{k,m} > g_{k,m}(n)$$

$$X^{(m)}(n) = \frac{1}{m} \sum_{k=m(n-1)+1}^{mt} X(n)$$

On peut voir $X^{(m)}$ comme une vue à différentes échelles du processus. Quand m augmente on *voit* le processus de plus en plus loin, ce qui signifie que l'on ne peut plus distinguer toutes les valeurs, mais seulement leurs moyennes sur un intervalle de taille m. L'étude de l'évolution des statistiques (en particulier la variance) du processus $X^{(m)}$ en fonction de m permet de mettre en évidence certaines caractéristiques du processus comme l'invariance d'échelle. Ce point sera discuté dans la section 2.3 (chapitre 2).

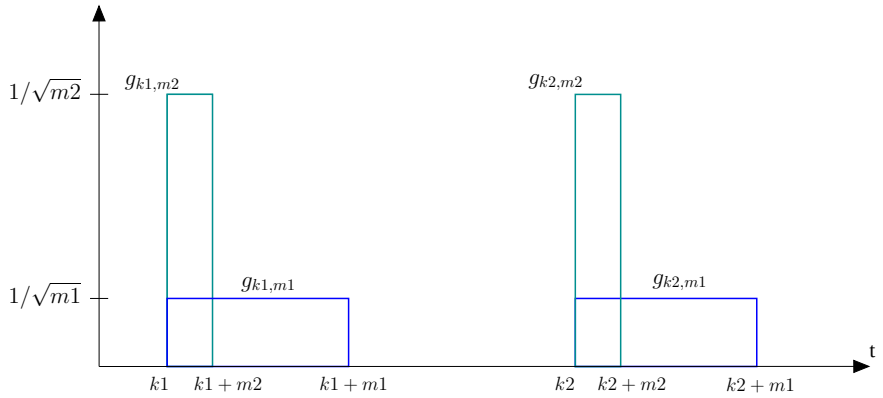


Fig. 1.1: La famille de fonction $g_{k,m}(n)_{k\in\mathbb{Z}}$ sur laquelle on projette X lorsque l'on calcule le processus agrégé $X^{(m)}$.

1.3 Modèles classiques

Les fondements et les propriétés principales des processus stochastiques étant posés, nous proposons dans cette section une revue de quelques modèles classiquement utilisés pour modéliser du trafic. L'objectif n'est pas ici d'en dresser une liste exhaustive, mais plutôt de motiver leur utilité dans le cadre de la modélisation de trafic. Chaque modèle sera décrit par ses statistiques d'ordre 1 (loi marginale) et d'ordre 2 (covariance).

1.3.1 Processus IID

Un processus stochastique IID (Indépendant Identiquement Distribué) $\{X[n]\}_{n\in\mathbb{N}}$ est une suite de variables aléatoires indépendantes et suivant toutes la même loi, c'est à dire ayant toute la même fonction de masse (PDF) :

$$\forall (n,m)\in\mathbb{N}^2,\quad f_{X[n]} = f_{X[m]} \stackrel{\Delta}{=} f_X$$

Les paramètres de ce modèle dépendent de la forme de la loi marginale choisie. Par exemple, un processus IID suivant une loi Normale (on dit alors qu'il est Gaussien car la loi Normale est aussi appelés loi de Gauss ou gaussienne) possède deux paramètres, la moyenne et la variance (voir section 2.10). La question du choix de la forme de loi marginale ainsi que celle de la méthode de détermination des paramètres, étant donnée une série, est discutée à la section 1.4.2. Ce modèle est simple, et il est aisé de synthétiser des réalisations de celui-ci à partir d'un générateur de nombre aléatoire (voir section 1.4.3).

De tels processus sont aussi appelé bruit blanc, à cause de la forme particulière de leur densité spectrale de puissance (droite horizontale). En effet, la covariance ne possédant qu'un point non nul ($\gamma(0) = \mathrm{Var}(X)$), il est aisé de montrer que $\Gamma(\nu) = \gamma(0)$, $\forall \nu$. Toutes les fréquences (ou couleurs par analogie avec la lumière) contribuant de la même manière à la variance du processus, on parle du bruit blanc, car c'est donc un mélange équilibré de toutes les fréquences (toutes les couleurs).

Les processus IID (qui sont aussi appelés *innovations*) sont beaucoup utilisés et comme cela sera détaillé dans les sections suivantes, ils sont souvent à la base de la définition d'autres processus.

Loi marginale La loi marginale des processus IID peut être quelconque.

Covariance L'indépendance des variables aléatoires entraîne que la covariance $\gamma_X(k)$ est nulle pour tout $k \neq 0$.

1.3.2 Processus ARMA

Un processus stochastique ARMA (pour *Auto Regressive Moving Average*) peut être vu comme le résultat du *filtrage* d'un processus IID gaussien noté ϵ. Le processus $\{\epsilon[n]\}_{n\in\mathbb{N}}$ est appelé *innovation*. Lorsque l'on applique un filtrage sur le processus IID, on modifie son spectre (qui n'est plus blanc alors), et on modifie ainsi la covariance du processus. Les dépendances introduites sont dites à courte mémoire, car on peut montrer que la covariance décroît exponentiellement (voir chapitre 2). Ces processus ont fait l'objet de nombreux travaux, et ils sont utilisés dans de très nombreux domaines [25] (traitement du signal audio, des images, étude de séries financières, etc.).

Le processus $\{X[n]\}_{n\in\mathbb{N}}$ est un processus ARMA de paramètre Φ_p et Θ_q (on note souvent ARMA(p,q)) si :

$$\Phi_p(B)X[n] = \Theta_q(B)\epsilon[n] \tag{1.8}$$

avec :

- B : Opérateur de retard : $BX[n] = X[n-1]$, $B^2X[n] = X[n-2], \ldots$
- Φ_p : Polynôme de degré p : $\Phi_p(B) = 1 - \phi_1 B - \phi_2 B^2 \ldots - \phi_p B^p$. Ce polynôme correspond à la partie moyenne mobile (MA pour *Moving Average*) du processus.
- Θ_q : Polynôme de degré q : $\Theta_q(B) = 1 - \theta_1 B - \theta_2 B^2 \ldots - \theta_q B^p$. Ce polynôme correspond à la partie auto-régressive (AR pour *Auto Regressive*) du processus.
- $\{\epsilon[n]\}_{n\in\mathbb{N}}$: Innovations, processus IID gaussien de moyenne nulle et de variance σ^2

On peut montrer que la densité spectrale de puissance des processus ARMA est telle que :

$$\Gamma_X(\nu) = \frac{\sigma^2}{2\pi} \frac{|\Phi_p(e^{-2i\pi\nu})|^2}{|\Theta_q(e^{2i\pi\nu})|^2}$$

Ceci implique [40] que la covariance soit au moins exponentiellement décroissante, c'est à dire qu'il existe $s \in [-11]$, tel que :

$$\gamma_X(k) \leq cs^{|k|} = ce^{-|k|log\frac{1}{s}}, \quad c \in \mathbb{R}^*$$

Remarque : La covariance ne s'exprime pas par une formule analytique, néanmoins il est possible de la calculer numériquement [40, 25].

Marginale : La loi marginale d'un processus ARMA dépend uniquement de la loi de probabilité suivie par les variables aléatoires $\epsilon[n]$ (l'innovation). Si cette loi est une loi Normale, alors on dit que c'est un processus ARMA gaussien.

Covariance La forme de la covariance est contrôlée dans le domaine fréquentiel par les polynômes Φ_p et Θ_q. De part la décroissance exponentielle de la covariance, les processus ARMA sont dits à courte mémoire.

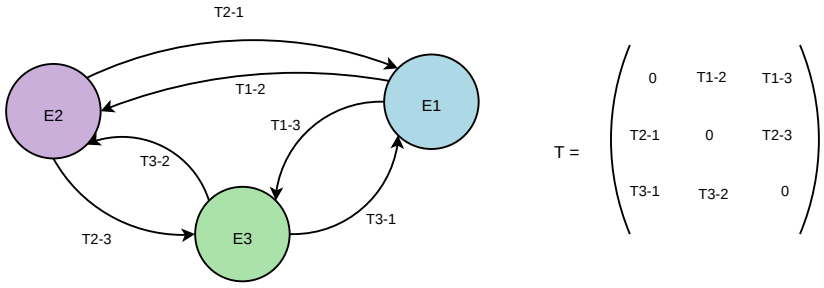


Fig. 1.2: Exemple de chaîne de Markov à 3 états et la matrice de transition correspondante.

1.3.3 Processus Markoviens

Nous présentons dans cette section les processus Markoviens. Nous n'avons pas directement utilisé ces processus dans nos travaux, néanmoins ils sont présentés ici à titre d'état de l'art car ces processus ont été considérablement utilisés pour la modélisation de trafic [120, 162, 136, 109, 6]. Nous nous sommes en effet surtout intéressés à la caractéristique de longue mémoire, qui est difficile à intégrer dans ces modèles. Les chaînes de Markov seront introduites dans la section 1.3.3.1, puis les processus de Poisson seront présentés dans la section 1.3.3.2. Enfin la combinaison de ces derniers, c'est à dire les processus de Poissons modulés par une chaîne de Markov (MMPP) seront décrits dans la section 1.3.3.3.

1.3.3.1 Chaînes de Markov à temps discret

Une manière intuitive de présenter une chaîne de Markov à temps discret est de la représenter comme un graphe où les sommets sont des *états* (l'ensemble des états est noté $\Sigma = \{E_k\}_{k\in I}$) et les arcs des *transitions* comme cela est illustré sur la figure 1.2. L'ensemble Σ doit être dénombrable. A chaque transition de l'état E_i à l'état E_j, on associe une valeur qui correspond à la probabilité de passer de l'état E_i à l'état E_j. On construit ainsi une *matrice stochastique*[1] de transition, carrée de taille $card(\Sigma)$, notée T. On associe à la chaîne de Markov un processus stochastique $\{X[n]\}_{n\in\mathbb{N}}$, la valeur de $X[n]$ correspondant à l'état dans lequel on se trouve à l'instant n. La propriété fondamentale des chaînes de Markov est qu'elles sont *sans mémoire* : $\forall\, j, E_j \in \Sigma,\ \forall n$,

$$\mathbb{P}\Big(X[n+1] = E_{n+1} | X(0) = E_0, \ldots, X[n] = E_n\Big) = P\Big(X[n+1] = E_{n+1} | X[n] = E_n\Big) \tag{1.9}$$

L'évolution de la chaîne de Markov est regardée au travers d'un vecteur noté $\pi[n]$, de taille $card(\Sigma)$ et qui contient les probabilités d'être dans chacun des états à l'étape n. On a par définition :

$$\pi[n+1] = \pi[n]\,T \tag{1.10}$$

$\pi(0)$ représente les probabilités initiales d'être dans chacun des états de Σ. On peut montrer [111] que sous certaines conditions (la chaîne doit être apériodique et tous les états doivent être récurrents non nuls), $\pi(n)$ converge vers $\pi = \lim_{n\to\infty}\pi(n)$. Ce vecteur π représente les probabilités stationnaires d'être dans chacun des états, c'est à dire la loi marginale du processus X.

Les chaînes de Markov sont utilisées pour modéliser une grande variété de systèmes, des plus simples aux plus complexes. Elles sont souvent construites à partir d'une modélisation *comportementale* d'un système. En revanche elles ne sont pas directement utilisées pour générer du

[1] Une matrice stochastique possède la propriété que chaque somme des éléments d'une ligne est égale à 1.

trafic. Il faudrait pour cela construire une chaîne de Markov ayant autant d'états que de valeurs possibles voulues. Il faudrait ensuite déterminer les probabilités de transitions, c'est à dire les probabilités que le processus passe de la valeur E_i à la valeur E_j. Cela fait un nombre très important de paramètres qu'il faut extraire des données, et c'est pourquoi ces processus ne sont pas en pratique directement utilisés pour modéliser du trafic. On peut aussi montrer que la propriété "sans mémoire" impose une fonction de covariance exponentiellement décroissante [111]. Néanmoins, on peut utiliser les chaînes de Markov à temps discret pour *moduler* l'intensité d'un processus de Poisson. Cette approche permet d'obtenir un processus ayant des caractéristiques intéressantes comme cela va être montré dans les deux sections suivantes.

1.3.3.2 Processus de Poisson

Un processus ponctuel caractérise les arrivées d'événements dans un système au cours du temps. Un processus de Poisson est un processus ponctuel pour lequel la distribution du nombre d'arrivées dans le système dans un intervalle de temps T noté $N(T)$ suit, pour tout intervalle T, une loi de Poisson de paramètre λT :

$$\forall T > 0, \forall k \in \mathbb{N}, \quad \mathbb{P}(N(T) = k) = e^{-\lambda T} \frac{(\lambda T)^k}{k!} \tag{1.11}$$

On peut montrer que dans ces conditions le temps entre deux arrivées suit une loi exponentielle de paramètre $\mu = 1/\lambda$.

Ces processus sont très utilisés car ils modélisent très bien le nombre d'arrivées de clients dans un système (le nombre d'appels reçus par un standard téléphonique par exemple). On peut aussi les utiliser, et cela a été fait pendant longtemps, pour modéliser l'arrivée des paquets sur les liens d'un réseau de machines, et caractériser ainsi par exemple le trafic Internet. Comme cela sera détaillé dans la section 2.2, ce type de modélisation n'est plus utilisé à l'heure actuelle car elle ne rend pas bien compte des caractéristiques du trafic Internet [121], et c'est pourquoi les processus de Poisson modulés par une chaîne de Markov décrits dans la section suivante ont été introduits.

1.3.3.3 MMPP

Afin d'avoir de pouvoir s'adapter à une large gamme de situations, on peut utiliser un processus de Poisson dont l'intensité est modulée par une chaîne de Markov. L'intensité (λ) est alors fonction de l'état d'une chaîne de Markov. Ces processus sont regroupés sous l'acronyme MMPP (pour *Markov Modulated Poisson Process*) [51]. Un $M-$MMPP est un MMPP à M états.

La loi marginale d'un MMPP est une combinaison de lois de Poisson, et sa covariance est une somme pondérée d'exponentielles. Les paramètres du modèle sont nombreux puisqu'en plus de la matrice de transition de la chaîne de Markov, une nouvelle matrice diagonale notée Λ, contenant les intensités associées aux différents états est nécessaire.

Le modèle ON/OFF est le plus simples des MMPP. La chaîne de Markov modulant le processus de Poisson possède deux états : ON et OFF, qui correspondent à deux valeurs de l'intensité d'un processus de Poisson (on parle de modèle ON/OFF car souvent dans l'état off l'intensité est nulle). En plus de la matrice 2x2 de transition, ce modèle nécessite 2 nouveaux paramètres : λ_{on} et λ_{off}, qui correspondent aux intensités respectives du processus de Poisson dans les états ON et OFF (voir figure 1.3). Ce modèle possède donc 4 paramètres (p,q,λ_{on} et λ_{off}). On peut

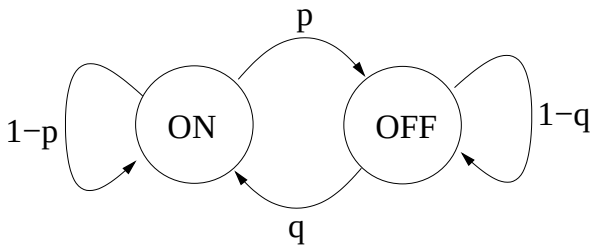


Fig. 1.3: Modèle ON/OFF. Une chaîne de Markov à deux états module l'intensité d'un processus de Poisson.

facilement montrer que la chaîne de Markov associée à un processus ON/OFF possède un état stationnaire [111] $\pi = (\pi_{on}, \pi_{off}) = (\frac{p}{p+q}, \frac{q}{p+q})$.

La loi marginale du processus ainsi obtenu est une mixture de deux lois de Poisson de paramètres λ_{on} et λ_{off}. Sa loi marginale est donc :

$$f(k) = \pi_1 g_{\lambda_{on}}(k) + \pi_2 g_{\lambda_{off}}(k), \qquad k = 0,1,2,\ldots \qquad g_\lambda(k) = e^{-\lambda}\frac{\lambda^k}{k!}$$

et sa covariance est :

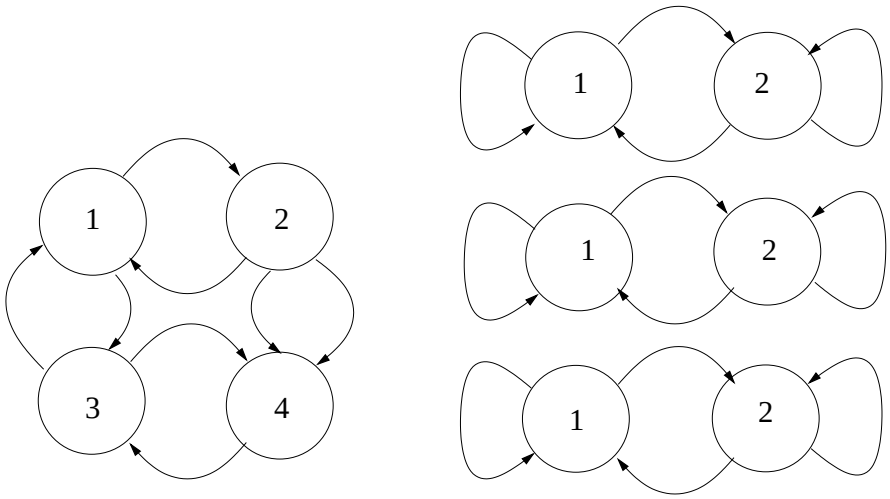
$$f(k) = \pi_1\pi_2|\lambda_{off} - \lambda_{on}|^2 e^{kc_l}, \quad c_l = \ln(1-p-q), \qquad k = 0,1,2,\ldots$$

Les modèles ON/OFF sont des modèles simples permettant de modéliser l'alternance de deux phases ou modes de communications ayant des caractéristiques différentes. Ceci permet de reproduire une certaine intermittence, qui est une caractéristique très importante du trafic car elle possède un impact important sur les performances (voir section 2.2).

De nombreux modèles à base de MMPP ont été proposés pour modéliser le télé-trafic [120, 162, 136, 109, 6]. Le travail de Salvador *et al.* [120] propose par exemple une méthodologie pour arriver à déterminer les paramètres du modèle pour un ensemble d'observations donné. L'idée est de trouver les meilleurs paramètres d'un MMPP afin de reproduire au mieux la loi marginale et la covariance de cet ensemble. En pratique, il est difficile de chercher à bien reproduire la marginale et la structure de corrélations de manière indépendante car tous les paramètres du modèle (T, λ_{on} et λ_{off}) influencent les deux en même temps.

Pour contourner ce problème, il faut utiliser une propriété des MMPP qui est que la superposition de deux MMPP est aussi un MMPP dont les paramètres se dérivent facilement [51]. On va donc chercher à construire deux MMPP, un reproduisant la loi marginale, et un autre la covariance du processus (voir figure 1.4).

- Le MMPP en charge de reproduire la covariance est une superposition de L 2−MMPP (ON/OFF). La covariance d'un 2−MMPP étant une exponentielle, la covariance résultante est une somme pondérée de L fonctions exponentielles de fréquences différentes. Il faudra donc modéliser la covariance des données par une somme pondérée d'exponentielles (des algorithmes existent pour effectuer cette tâche [49]).
- Le MMPP en charge de reproduire la loi marginale du processus est un M−MMPP. Celui-ci doit avoir une covariance identiquement nulle pour ne pas interférer avec la covariance introduite par les L 2−MMPP, qui est la covariance finale que l'on souhaite obtenir. On peut montrer qu'il faut, pour cela, que toutes les lignes de sa matrice de transition soient identiques, et cela impose des restrictions sur les formes de lois marginales possibles [120].



4−MMPP − Loi marginale 3 2−MMPP − Covariance

Fig. 1.4: Un exemple de modèle étant la composition de trois 2-MMPP pour reproduire la structure de corrélation, et d'un 4-MMPP pour reproduire la loi marginale.

Une fois que tous les paramètres des différents MMPP du modèle sont estimés, on obtient le MMPP final en les *sommant* (cela revient à sommer des matrices de dimensions différentes, et il faut pour cela utiliser la somme de Kronecker [120]). On obtient alors un $(2L+M)$−MMPP qui peut être simulé pour produire des réalisations synthétiques de trafic.

Il est possible d'obtenir les expressions analytiques de la covariance du processus, ainsi que de sa loi marginale et dans [120], une méthodologie pour déterminer automatiquement les paramètres du modèle est proposée.

1.4 Modélisation d'un ensemble d'observations

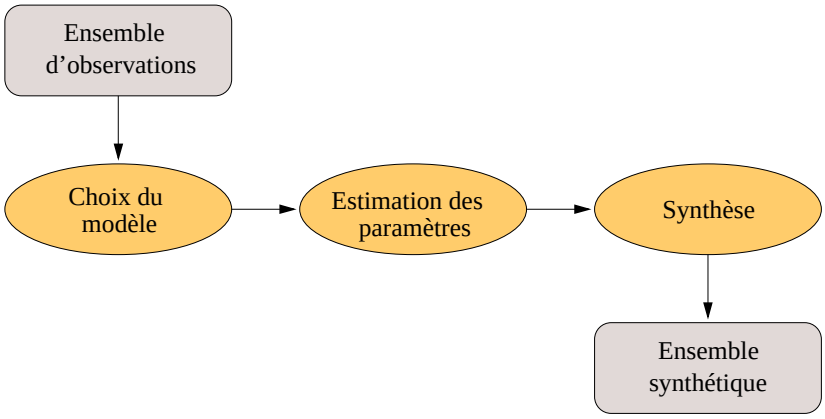


Fig. 1.5: Étapes de la modélisation d'un ensemble d'observations.

Dans cette section, nous présentons une revue rapide des différentes étapes qu'implique la *modélisation* d'un ensemble d'observations par des processus stochastiques, en vue d'une génération d'un ensemble synthétique, statistiquement proche de l'ensemble initial. Cela inclut,

comme indiqué sur la figure 1.5, la sélection d'un modèle (section 1.4.1), l'estimation des paramètres du modèle (section 1.4.2) et la génération d'une série synthétique (section 1.4.3). Nous ne détaillerons pas dans cette section la problématique de l'obtention des données, qui est spécifique à chaque domaine d'applications. Cette section présente des généralités plutôt qu'une description détaillée des moyens que nous avons mis en place pour la modélisation de trafic Internet et de trafic sur puce.

1.4.1 Sélection du modèle

La première étape de la modélisation est le choix d'un *modèle* de processus stochastique. Certains d'entre-eux ont été présentés dans la section 1.3, et d'autres seront décrits dans le chapitre 2. Un modèle consiste en général en une forme mathématique paramétrique des propriétés statistiques d'un processus stochastique (loi marginale, covariance, etc.). La sélection d'un modèle consiste à identifier quelles formes seront le mieux adaptées à un ensemble d'observations donné, soit à répondre à la question : quel modèle est le plus adapté pour cet ensemble d'observations ?

Ce choix implique de pouvoir comparer les différents modèles entre eux, et pour cela il faut définir une quantité permettant d'évaluer le *score* d'un modèle étant donné l'ensemble d'observations [27]. Différents types de scores existent, comme le critère d'information bayésien (BIC pour *Bayesian Information Criterion*) qui sera utilisé dans la section 6.2.3 (page 128) pour la comparaison de différents modèles de segmentation de trace de trafic sur puce.

En pratique, nous avons effectué cette étape manuellement, à partir de notre expérience et de l'analyse des propriétés statistiques des données que nous avons analysées (représentation de la loi marginale et de la covariance). Il faut, pour mettre en place un algorithme de sélection de modèle automatique, avoir une connaissance approfondie du type de donnée que l'on va analyser afin de pouvoir avoir confiance dans l'algorithme de sélection.

1.4.2 Estimation des paramètres et tests statistiques

Ayant choisi un modèle ayant un certain nombre de paramètres, il convient d'effectuer l'estimation de ses paramètres, c'est à dire trouver les meilleures valeurs de ceux-ci étant donné un ensemble de N observations (noté ici $\mathbf{x} = (x_1, \ldots, x_N)$). Ceci est une étape cruciale de la modélisation. C'est un problème à la fois théorique (mise au point et évaluation de la rapidité de convergence des estimateurs) et pratique (développement d'algorithmes rapides) qui a reçu une grande attention [25, 96, 67]. De nombreuses techniques d'estimation de paramètres ont donc été mises au point (méthode des moindres carrés, méthode de Gauss-Newton, maximum de vraisemblance, maximisation de l'espérance).

Ces méthodes cherchent toutes, dans l'espace Θ des paramètres (par exemple, pour un modèle de processus IID gaussien, l'espace est $(\mu, \sigma) = \mathbb{R}^2$), un point θ minimisant une fonction de coût qui caractérise la distance entre le modèle avec les paramètres θ et l'ensemble d'observations. La fonction de coût utilisée peut être la fonction de vraisemblance notée $L(\theta|\mathbf{x})$ (L vient du mot anglais *likelihood*). $L(\theta|\mathbf{x})$ est une fonction de p variable, p étant le nombre de paramètres du modèle, et dépend de l'ensemble d'observations $\mathbf{x}$. Dans le cas de l'estimation des paramètres d'une loi marginale, on introduit aussi la notation $f(x|\theta)$ qui est la fonction de masse PDF prise au point x, et qui indique que cette fonction dépend aussi des paramètres θ.

Si on fait l'hypothèse que les observations sont IID (dans ce cas on peut considérer que la loi

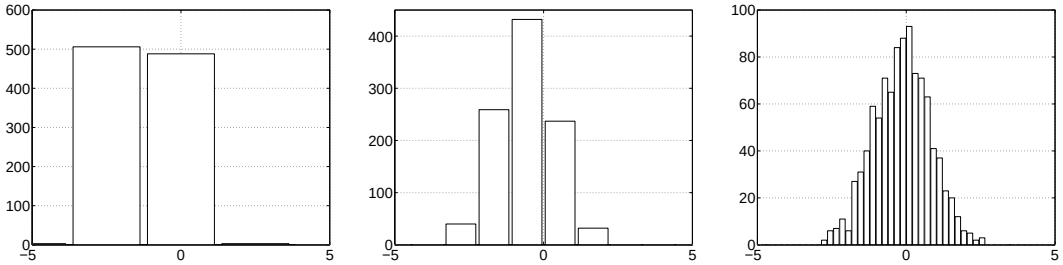


Fig. 1.6: Nombre d'apparitions par intervalles des valeurs d'un processus IID gaussien (1000 échantillons), pour différents nombres d'intervalles de même taille entre -5 et 5 (5, 10 et 50, de gauche à droite).

jointe est le produit des lois, voir section 1.1.2), alors on peut calculer analytiquement la fonction L :

$$L(\theta|\mathbf{x}) = \prod_{i=0}^{N} f(x_i|\theta)$$

La présence du produit est la raison pour laquelle on utilise plus souvent la *log-vraisemblance*. En effet, la fonction logarithme étant monotone, le point θ maximisant $L(\theta|\mathbf{x})$ maximise aussi $\log(L(\theta|\mathbf{x}))$, et on a :

$$\log(L(\theta|\mathbf{x})) = \sum_{i=0}^{N} \log(f(x_i,\theta))$$

L'estimateur de maximum de vraisemblance (MLE pour *Maximum Likelyhood Estimator*) est la valeur de θ qui maximise $L(\theta)$. Cette valeur peut être calculée analytiquement dans certains cas simples, si ce n'est pas possible il faut utiliser un algorithme d'optimisation pour trouver ce maximum. On peut par exemple montrer que dans le cas gaussien, l'estimation par maximum de vraisemblance des paramètres donne :

- $\mu = \frac{1}{N}\sum_{i=0}^{N} x_i \triangleq \hat{\mu}$
- $\sigma^2 = \frac{1}{N}\sum_{i=0}^{N} x_i^2 - \hat{\mu}^2 \triangleq \hat{\sigma}^2$

$\hat{\mu}$ et $\hat{\sigma}^2$ sont appelés respectivement estimateurs de moyenne et de variance standards [67].

Il est aussi intéressant, afin de pouvoir comparer les modèles et vérifier qu'une estimation est valide, d'avoir un indicateur de la qualité d'une estimation (*goodness-of-fit* en anglais). Ces indicateurs prennent souvent la forme de tests statistiques visant à décider si un certain modèle est pertinent pour l'ensemble d'observations. Les tests de Kolmogorov-Smirnov ou de Anderson permettent par exemple d'avoir un indicateur de la qualité d'une forme de loi marginale sur des données à valeurs continues [67]. Le test du chi-deux (χ^2) est plus adapté aux observations ayant des valeurs discrètes et nous l'utiliserons pour tester l'hypothèse qu'une loi marginale observée dans un ensemble d'observations suit une loi donnée notée $\mathscr{L}$ (Normale, Gamma, etc.).

Le test du χ^2 est formulé de la manière suivante pour un couple (données/modèle) : on définit l'hypothèse H_0, signifiant "les données suivent la loi $\mathscr{L}$", et le test permet de décider si on doit rejeter ou accepter H_0. Cette décision est basée sur le calcul d'une statistique notée S. Pour la calculer, il faut choisir un ensemble de M intervalles disjoints dans $\mathbb{R}$. Pour chaque intervalle i, on va compter le nombre de valeurs des observations qui y sont comprises ($F_d(i)$). On obtient

intuitivement une représentation grossière de la PDF comme le montre la figure 1.6 pour différents nombres d'intervalles. On appelle ces représentations des *histogrammes empiriques* (voir section 1.1.2) et ce terme sera régulièrement utilisé dans la suite du texte pour s'y référer. On doit ensuite calculer, le nombre de valeurs théoriquement comprises dans chaque intervalle ($F_t(i)$) à partir de l'expression de la CDF de la loi $\mathscr{L}$. La statistique S est alors définie ainsi :

$$S = \sum_{k=0}^{M} \frac{F_d(k) - F_t(k)}{F_t(k)} \tag{1.12}$$

Intuitivement, plus S sera petite, plus l'histogramme empirique sera proche de la forme de loi $\mathscr{L}$ (car les fréquences d'apparitions seront proches). On peut montrer que S suit une loi du χ^2 à $M-m$ degré de liberté, m étant égale au nombre de paramètres de la loi théorique moins 1. On peut ainsi fixer un intervalle de confiance (typiquement 1%, 5% ou 10% en pratique) et calculer la limite supérieure de S, au-delà de laquelle on rejettera l'hypothèse H_0 selon laquelle les observations suivent la loi $\mathscr{L}$. Fixer l'intervalle de confiance revient à fixer le pourcentage de chance que l'on a de se tromper en acceptant l'hypothèse H_0. On peut ainsi décider automatiquement si un ensemble d'observations suit ou non une loi $\mathscr{L}$ donnée. On peut aussi calculer une valeur, notée P, définit comme 1 moins la valeur de la distribution cumulative d'une loi du chi^2 à $M-m$ degré de liberté au point S. Cette valeur donne un indicateur de confiance dans l'acceptation de l'hypothèse H_0. En particulier, si P vaut 1, alors l'hypothèse H_0 ne fait aucun doute, si P vaut 0, au contraire l'hypothèse H_0 ne peut jamais être acceptée, et si P vaut par exemple 10%, alors l'hypothèse sera admise avec un intervalle de confiance de 10%.

1.4.3 Synthèse de processus

Cette section présente brièvement différents algorithmes permettant d'obtenir des réalisations de processus IID. Tout procédé de génération de réalisation de processus stochastique contient en effet une étape de génération de processus IID, c'est pourquoi cette problématique est rapidement introduite ici.

Synthétiser un processus IID revient à générer, à l'aide d'un algorithme, une suite de nombres aléatoires suivant une loi marginale donnée. Ceci n'est pas une chose facile étant donnée la nature déterministe du comportement des algorithmes. Il est impossible de générer une suite de nombres réellement aléatoires. On peut néanmoins générer une suite dite *pseudo-aléatoire*, c'est à dire une suite dont on peut considérer, malgré le fait qu'elle soit issue d'une construction déterministe, qu'elle a un comportement aléatoire. Une revue des différentes méthodes de génération ainsi que des tests statistiques que l'on peut effectuer pour tester la *qualité* de l'aléatoire produit est disponible dans [79].

En général les générateurs de nombres aléatoires suivant une loi donnée reposent sur un générateur de nombres uniformément répartis [83, 84], c'est à dire d'un générateur de réalisations d'une variable aléatoire X suivant une loi uniforme ($\mathbb{P}(X = k) = 1/(b-a), k \in [a,b]$, voir section 2.10). En effet, si X est uniformément répartie entre 0 et 1, alors la variable aléatoire $Y = F^{-1}(X)$ aura la fonction F pour fonction de répartition (CDF) [55]. Une bonne revue des différents algorithmes permettant de générer des séquences de nombres aléatoirement répartis suivant différentes lois a été réalisé par l'Ecuyer [83, 84].

La génération de processus IID est un problème déjà bien étudié et dont les principaux résultats sont depuis longtemps posés [83]. Nous nous sommes reposés sur des générateurs existants pour effectuer cette tâche.

La synthèse de réalisation de processus ARMA est aussi un problème largement étudié, le lecteur intéressé est renvoyé à l'article [25], qui fait une revue des différentes techniques existantes.

1.5 Conclusion

Ce premier chapitre nous a permis de poser les bases théoriques de la modélisation d'un ensemble d'observations à l'aide de processus stochastiques. Ce domaine de recherche est très actif et pluridisciplinaire par nature puisque les observations peuvent provenir d'expériences ou de simulations dans tous les domaines de la science. Il n'existe néanmoins pas de modélisation miracle, fonctionnant pour n'importe quelles données, la modélisation doit au contraire être faite avec une bonne connaissance des données à analyser, et dépend de l'utilisation que l'on souhaite faire de cette modélisation. Il existe un grand nombre de modèles, allant du plus simple (processus IID) au plus complexe, mais la complexité du modèle n'implique pas forcément que la modélisation réalisée sera de meilleure qualité. Par exemple, nous nous sommes limiter, dans ce chapitre, aux processus stationnaires, et si les observations que l'on souhaite modéliser montrent une importante non-stationnarité, alors peu importe la complexité du modèle et des procédures d'estimation de paramètres mis en jeu, la modélisation n'aura pas de sens. Dans ce cas, le modèle n'étant pas adapté aux observations, l'utilisation qui sera faite de la modélisation sera erronée.

Dans le chapitre suivant, nous présentons une propriété de certains processus stochastiques, la longue mémoire, qui est réputée pour avoir un impact important pour les performances des réseaux.

Chapitre 2
Longue mémoire

Ce chapitre présente les fondements mathématiques de la longue mémoire et de l'autosimilarité ainsi que quelques processus classiques utilisés pour modéliser ces caractéristiques. La longue mémoire est une caractéristique importante qui a été identifiée dans de nombreuses données (trafic Internet [121, 118], biologie, etc.). Elle est particulièrement importante dans le trafic, car elle possède un impact négatif sur les performances des réseaux [118]. En particulier, les performances des éléments de mémorisation à l'entrée des routeurs (*buffers*) sont largement différentes de celles obtenues avec un trafic n'ayant pas cette propriété. C'est d'ailleurs cet impact sur les performances du réseau, ainsi que la volonté de faire une modélisation précise des caractéristiques statistiques du trafic, qui ont guidé les premières études de cette caractéristique dans le cadre du trafic Internet [121]. Cette propriété est maintenant communément admise dans le trafic Internet, et il convient d'en tenir compte dans une tentative de modélisation de celui-ci.

L'étude des processus stochastiques à longue mémoire a reçu une attention considérable dans la communauté scientifique ces dix dernières années (voir la revue [157] et les livres [40, 118] pour une bibliographie orientée trafic des travaux sur la longue mémoire).

Ce chapitre est organisé comme suit : la longue mémoire est introduite et définie dans la section 2.1, et l'impact de cette propriété sur les performances est discuté dans la section 2.2. L'autosimilarité est introduite dans la section 2.3 et la relation entre longue mémoire et autosimilarité est ensuite clarifiée dans la section 2.5. Une discussion sur le sens du paramètre de longue mémoire est ensuite menée dans la section 2.4 et la famille de processus FARIMA est présentée en détail dans la section 2.6. Différentes techniques d'estimation du paramètre de longue mémoire sont alors présentées dans la section 2.7 et la section 2.8 contient quant à elle une revue de différents algorithmes pour synthétiser des réalisations de processus gaussiens à longue mémoire. Enfin pour clôturer l'état de l'art de cette première partie, un récapitulatif des différents processus introduits est proposé dans la section 2.10.

2.1 Définitions

Un processus $\{X[n]\}_{n\in\mathbb{N}}$ est dit à longue mémoire (LRD pour *Long Range Dependent*) si sa fonction d'auto-covariance $\gamma_X(k)$ n'est pas sommable :

$$\sum_{k\in\mathbb{N}} \gamma_X(k) = \infty \tag{2.1}$$

Ceci s'oppose aux processus à mémoire courte. Par exemple, ceux qui possèdent une covariance exponentiellement décroissante (donc sommable) comme les processus ARMA et *a fortiori* les processus IID. La longue mémoire implique que les données sont corrélées sur une période infiniment grande, c'est à dire que la corrélation entre deux variables aléatoires $X[n]$ et $X[m]$ du processus arbitrairement éloignées ($|n-m|$ aussi grand que l'on veut), ne peut pas être

négligée. Il n'existe pas de temps caractéristique à partir duquel on peut considérer que les *VA* du processus ne sont plus corrélées.

Les fonctions classiquement utilisées pour modéliser ce comportement sont les fonctions puissance ($f(x) = |x|^{-\alpha}, 0 < \alpha \leq 1$) car elles sont décroissantes vers 0 et que leurs sommes divergent. On modélise donc souvent un comportement à longue mémoire par une covariance $\gamma_X(k)$ comme suit :

$$\gamma_X(k) \underset{k\to+\infty}{\sim} c\,|k|^{-\beta}, \quad 0 < \beta \leq 1, \quad c \in \mathbb{R}^*$$

La propriété de longue mémoire peut aussi être vue dans la densité spectrale de puissance (qui est la transformée de Fourier de la fonction de covariance, voir section 1.2.5). Le fait que la somme de la covariance diverge implique en effet que sa représentation fréquentielle (la densité spectrale de puissance, voir section 1.2.5, page 18) diverge à l'origine des fréquences, et on peut facilement montrer que :

$$\sum_{k\in\mathbb{N}} \gamma_X(k) = \infty \quad \Rightarrow \quad \Gamma_X(0) = \infty$$

et :

$$\gamma_X(k) \underset{k\to+\infty}{\sim} c\,|k|^{-\alpha} \quad \Rightarrow \quad \Gamma_X(\nu) \underset{\nu\to 0}{\sim} c\,|\nu|^{-\alpha-1}, \quad \beta = 1-\alpha, \qquad 0 < \alpha \leq 1, \quad c \in \mathbb{R}^* \tag{2.2}$$

Les paramètres de longue mémoire d'un processus caractérisent la rapidité de décroissance de la covariance, ce sont donc au choix les exposants α ou β. C'est en général l'exposant α qui est utilisé. Il varie entre 0 et 1.

2.2 Longue mémoire et performance des réseaux

La longue mémoire est une propriété ayant un impact important sur les performances [118, 42, 78]. Les performances d'un réseau de télécommunications (utilisation des *buffers* dans les routeurs, rapidité de commutation, temps de séjour des paquets dans les routeurs, etc.) étaient jusqu'à il y a une dizaine d'années étudiées avec du trafic généré à partir de modèles à courte mémoire (IID, ARMA, MMPP, etc.). Cette modélisation a été faite à une époque ou le réseau Internet était naissant, à partir de modèles issus des réseaux téléphoniques (processus de Poisson). Il s'est avéré, lorsque les premières traces de trafic Internet ont été méticuleusement analysées, que les modèles à courte mémoire ne rendaient pas bien compte des caractéristiques statistiques [121] du trafic, et donc que l'évaluation de performance que l'on réalisait à partir de ces modèles était faussée.

Le problème majeur est que les performances du réseau sont dégradées en présence de longue mémoire. Par exemple, la distribution du remplissage d'une file d'attente (la probabilité d'avoir k clients dans la file, $k \in \mathbb{N}$) décroît beaucoup plus lentement si le processus d'arrivée est à longue mémoire (décroissance en loi de puissance) que si le processus n'est pas à longue mémoire (décroissance exponentielle, voir section 1.3.3.2). Ainsi la probabilité d'avoir un grand nombre de clients dans la file est élevée et pour un taux de perte donné, il faudra utiliser une file plus grande si le processus d'arrivée dans la file d'attente est à longue mémoire [118, 42].

Ainsi, si on ne tient pas compte de cette propriété et que la modélisation est utilisée pour dimensionner des éléments de réseaux, alors il y a un très grand risque pour que ce dimensionnement soit largement sous-évalué. Le cas simple du taux de remplissage d'une file d'attente lorsque le processus d'arrivée dans cette file est à longue mémoire a été étudié par Erramilli [42].

Une bonne revue des travaux concernant l'impact de la longue mémoire sur les performances est présentée dans le livre de Park *et al.* [118].

2.3 Autosimilarité

Cette section présente les fondements de l'autosimilarité ainsi que des processus possédant cette caractéristique. L'autosimilarité représente la propriété d'un processus d'être *statistiquement proche* de lui-même lorsqu'on le regarde à différentes échelles [118]. L'idée est que l'on ne peut pas trouver une échelle caractéristique pour lequel le processus peut être clairement caractérisé, toutes les échelles (sur une gamme d'échelles) sont équivalentes. *"Le tout ressemble à sa partie et la partie ressemble au tout"*. Cela est aussi appelé comportement fractal, et on peut d'ailleurs relier la *dimension fractale* [3] d'une réalisation d'un processus autosimilaire avec son autosimilarité. Une méthode intuitive pour comprendre l'autosimilarité d'un processus est d'imaginer des *zooms* successifs du processus. S'il n'est statistiquement pas possible de séparer ces différentes vues du processus, alors le processus est autosimilaire.

Formellement, on dit qu'un processus Y est autosimilaire de paramètre H (H_{ss} pour *Self Similar*, H est le paramètre de Hurst) si l'on le peut le "*dilater*" et obtenir un processus ayant les mêmes caractéristiques statistiques (mêmes lois conjointes finies) :

$$\forall a > 0, \forall n \in \mathbb{N}, \quad Y[n] \text{ possède la même loi que } a^{-H} Y[an], \quad H > 0 \tag{2.3}$$

On peut montrer qu'un tel processus Y ne peut pas être stationnaire. Nous allons donc considérer une sous-classe de ces processus ayant comme propriétés supplémentaires un processus des accroissements de Y noté X ($X[n] = Y[n] - Y[n-1]$) stationnaire, et une variance finie. On dit alors que Y est H_{sssi} (pour *Self Similar with Stationnary Increments*). Le processus $\{X[n]\}_{n\in\mathbb{N}}$ est quant à lui dit "*autosimilaire au second ordre*". Comme nous nous intéressons à la modélisation à l'aide de processus stationnaires, nous allons plutôt nous intéresser au processus X. On peut montrer, à partir des propriétés de Y, que le processus X possède les mêmes lois conjointes finies que le processus agrégé normalisé $m^{1-H}\{X^{(m)}[n]\}_{n\in\mathbb{N}}$. Si cela est vrai pour les grands m, alors le processus est dit "*asymptotiquement autosimilaire au second ordre*".

Le *mouvement brownien fractionnaire* (FBM pour *Fractionnal Brownian Motion*) est le processus le plus utilisé pour modéliser l'autosimilarité. Il est H_{sssi} et ses incréments sont gaussiens ce qui garanti que tous ses moments sont finis. Le processus de ses accroissements est appelé quant à lui *bruit gaussien fractionnaire* FGN pour *Fractionnal Gaussian Noise*), et de par sa stationnarité et sa longue mémoire (voir section suivante), c'est un processus très utilisé pour modéliser le trafic Internet.

On peut montrer que la covariance d'un FGN possède une forme particulière :

$$\gamma_X(k) = \frac{\sigma^2}{2}\left[(k+1)^{2H} - 2k^{2H} + (k+1)^{2H}\right], \qquad k \in \mathbb{N} \tag{2.4}$$

et un simple calcul montre :

$$\mathrm{Var}(X^{(m)}) = \sigma^2 m^{-\beta}, \qquad \beta = 2H - 2 \tag{2.5}$$

La variance du processus agrégé décroît donc moins vite que $\frac{1}{m}$ lorsque m augmente. Cette propriété est illustrée par la figure 2.1, qui montre le processus agrégé dilaté $m^{1-H}X^{(m)}$ pour différentes valeurs de m. On peut voir la différence entre un processus IID (pas de longue mémoire,

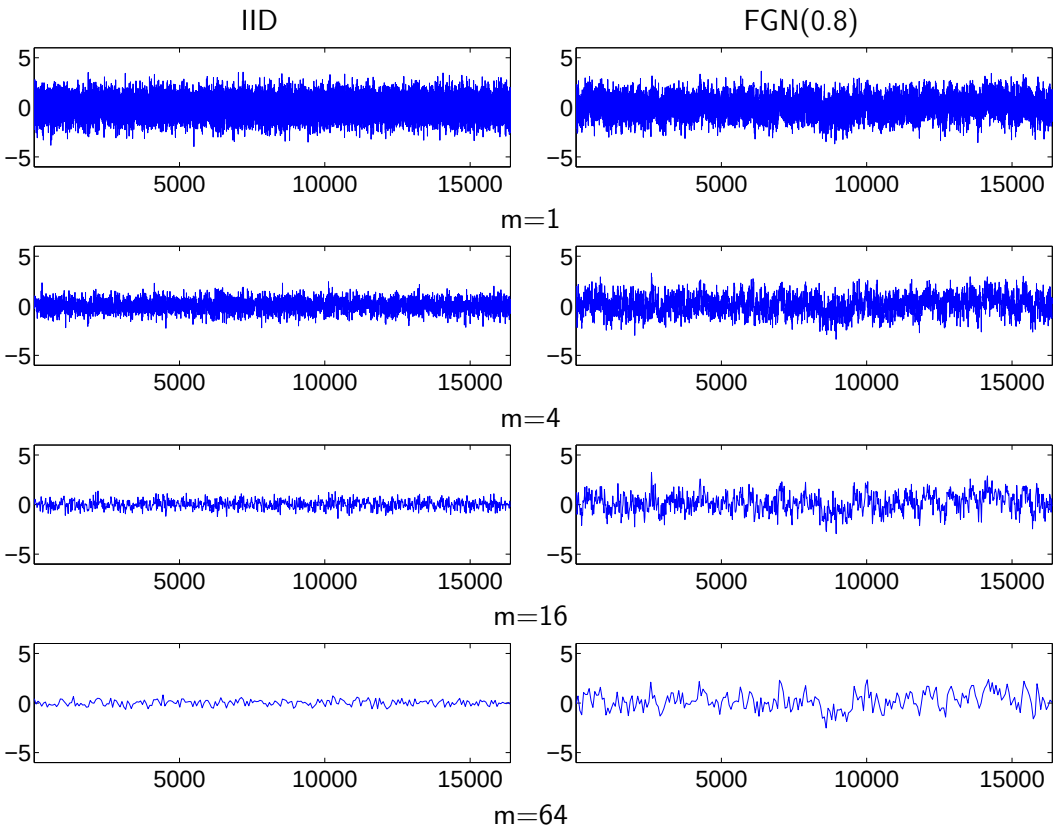


Fig. 2.1: Processus agrégé $X^{(m)}$ de la réalisation d'un processus IID (gauche) et d'un bruit gaussien fractionnaire de paramètre $H = 0.8$ (droite), pour $m = 1, 4, 16, 64$ (de haut en bas).

$H = 0.5$) et un processus à longue mémoire ($H = 0.8$ sur la figure). Dans le cas IID, la variance (variabilité) du processus agrégé décroît plus vite que dans le cas d'un processus à longue mémoire. On peut donc résumer cela en affirmant que plus un processus est à longue mémoire (plus H est proche de 1), plus la variance du processus agrégé associé décroît lentement. Cette propriété permet d'ailleurs d'estimer le paramètre de Hurst H comme cela sera montré dans la section 2.7.2.

2.4 Sens du paramètre de Hurst

Nous donnons ici quelques précisions sur le paramètre H pour un processus autosimilaire.

- Si $H = 1/2$, alors le processus est à mémoire courte (ARMA, IID, processus Markoviens).
- La décroissance de la covariance est d'autant plus lente que H est proche de 1. On dira que la mémoire est d'autant plus longue que H est proche de 1.
- Si on mesure un H supérieur à 1, alors on sort du cadre des processus à longue mémoire. Ceci peut aussi être la signature de non-stationnarité dans la trace.
- La valeur typiquement observée sur les traces de trafic Internet est située entre 0.7 et 0.8.

La figure 2.2 montre des bruits gaussiens fractionnaires (FGN) et des mouvements browniens fractionnaires (FBM, autosimilaire) pour différentes valeurs du paramètres de Hurst. Le paramètre de Hurst introduit dans le FGN une sorte d'intermittence (permettant de modéliser

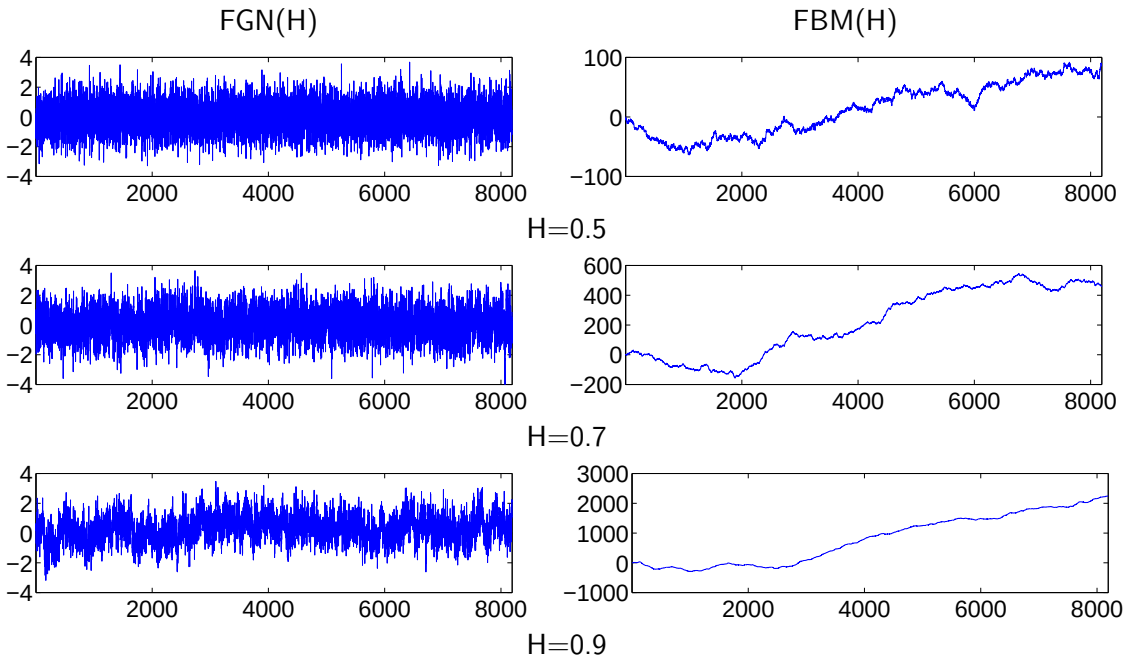


Fig. 2.2: Différents bruits gaussiens fractionnaires (gauche) et mouvements browniens fractionnaires (droite) pour différentes valeurs du paramètre H.

des *bursts*) d'autant plus forte que H est proche de 1. Pour le FBM, on peut relier le paramètre de Hurst à l'exposant de Hölder local (aussi appelé régularité locale) de la réalisation du processus. Cela revient à dire que la réalisation sera d'autant plus régulière que H est proche de 1. A la limite, quand $H = 1$, alors les réalisations du processus devraient être des fonctions continues (C^1). Le FBM est un processus qui a de nombreuses applications dans une large gamme de domaines scientifiques allant de la physique à la biologie en passant par la chimie. Néanmoins, comme nous nous intéressons à la modélisation de trafic, et que dans ce cadre il n'est pas directement adapté, nous allons nous focaliser sur l'analyse du processus de ces incréments (FGN) et des processus dérivés de celui-ci.

2.5 Lien entre longue mémoire et autosimilarité

On peut montrer qu'un processus asymptotiquement autosimilaire au second ordre possède une longue mémoire. Le FGN constitue donc une première grande famille de processus stationnaires à longue mémoire. On peut en effet montrer, à partir de l'expression de sa covariance, (équation (2.4)) que l'on a :

$$\gamma_X(k) \underset{k\to+\infty}{\sim} c|k|^{2H-2}$$

De par la définition de la longue mémoire (équation (2.2)), on peut relier les paramètres de longue mémoire au paramètre de Hurst :

$$-\beta = 2H-2 \Leftrightarrow \alpha - 1 = 2H-2 \Leftrightarrow \alpha = 2H-1 \Leftrightarrow H = \frac{\alpha+1}{2} = \frac{2-\beta}{2}$$

Cela est valable uniquement si le paramètre de Hurst est compris entre 1/2 et 1, pour respecter $0 < \alpha < 1$. La longue mémoire est donc à strictement parler équivalente à l'autosimilarité

asymptotique au second ordre, pour $0 < H < 1$. En raison de cette analogie, on utilise souvent le paramètre de Hurst pour la longue mémoire, c'est d'ailleurs ce que nous ferons dans la suite de ce texte. Les remarques relatives aux valeurs de H faite dans la section précédente sont donc applicables à la longue mémoire.

2.6 Processus FARIMA

Une autre famille de processus à longue mémoire est celle des processus FARIMA (pour *Fractionnaly Integrated Auto-regressive Moving Average*). Ils constituent une extension des processus ARMA (courte mémoire, voir section 1.3.2) en y introduisant une caractéristique de longue mémoire par intégration fractionnaire.

On peut différencier un processus $\{X[n]\}_{n\in\mathbb{N}}$ en utilisant l'opérateur noté $\Delta^d, d \in \mathbb{Q}$ défini ainsi comme suit (B est l'opérateur de retard introduit à la section 1.3.2) :

$$\Delta^0 X(n) = X[n], \quad \Delta^1 X[n] = X[n-1] - X[n], \ldots \quad \Delta^d X(n) = (I-B)^d X[n]$$

Dans le cas où d est fractionnaire, on utilise le développement suivant [40, 25] :

$$\forall j \in \mathbb{N}, \exists b_j, \quad \Delta^{-d} X[n] = (I-B)^{-d}\epsilon[n] = \left(\sum_{j=0}^{\infty} b_j B^j\right)\epsilon[n] = \sum_{j=0}^{\infty} b_j \epsilon[n-j]$$

On peut ainsi définir un processus ARMA "*intégré*", appelé ARIMA (pour *Integrated* ARMA) dans le cas ou d est entier et FARIMA (pour *Fractionnaly Integrated* ARMA) si d est fractionnaire. En utilisation les notations introduites lors de la définition de processus ARMA à la section 1.3.2, on a :

$$\Phi_p(B)\Delta^d X[n] = \Theta_q(B)\epsilon[n]$$

Ou, ce qui est équivalent et plus utilisé :

$$\Phi_p(B) X[n] = \Theta_q(B)\Delta^{-d}\epsilon[n] \tag{2.6}$$

Ceci revient à dire que le processus des accroissements d'ordre d du processus X est un processus ARMA(p,q). La densité spectrale de puissance de ces processus prend la forme suivante :

$$\Gamma(\nu) = \frac{\sigma^2}{2\pi}\frac{|\Phi(e^{-2i\pi\nu})|^2}{|\Theta(e^{2i\pi\nu})|^2}|(1-e^{2i\pi\nu})|^{-2d} \tag{2.7}$$

La covariance d'un tel processus ne s'exprime pas par une formule analytique, on peut l'écrire comme transformée de Fourier inverse de la densité spectrale de puissance (2.7). Il existe différents algorithmes pour la calculer numériquement [40, 25].

On peut néanmoins montrer que si d est compris entre 0 et 1/2, alors le processus FARIMA possède une longue mémoire de paramètre $H = d + 1/2$. En effet, dans la limite des basses fréquences (grandes échelles), c'est le terme de droite ($|(1-e^{2i\pi\nu})|^{-2d}$) qui va dominer la densité spectrale de puissance, et on a :

$$\Gamma(\nu) \underset{\nu\to 0}{\sim} c|\nu|^{-2d}$$

En utilisant la définition de la longue mémoire (équation (2.2)), alors on a $\alpha = 2d$, ce qui revient à $H = d + 1/2$.

Le comportement dans les hautes fréquences (petites échelles) est contrôlé, comme pour un processus ARMA, par les paramètres Φ_p et Θ_q. Ceci sera illustré et discuté à la section 2.10, par la figure 2.5. En général, on note FARIMA(p,d,q), p et q étant les degrés des polynômes Φ_p et Θ_q, et d étant le paramètre de longue mémoire ($H = d + 1/2$). Quand $p = q = 1$, alors on note FARIMA(ϕ,d,θ), puisque les polynômes n'ont qu'un coefficient chacun.

Cette classe de processus va particulièrement nous intéresser, d'une part, parce qu'elle englobe les processus ARMA et ARIMA (en prenant $d = 0$, ou d entier) et d'autre part, car elle permet de modéliser une covariance complexe (avec des paramètres distincts pour contrôler la covariance aux petites échelles et aux grandes échelles). Nous utiliserons en particulier cette classe de processus pour la modélisation de trafic Internet présentée dans la section 4.1.

2.7 Estimation de la longue mémoire

Nous allons présenter ici trois méthodes permettant d'estimer le paramètre de Hurst H : la méthode R/S (section 2.7.1), l'analyse temps/variance (section 2.7.2) l'analyse en ondelette (section 2.7.3) qui est l'estimateur que nous avons utilisé. Il existe d'autres estimateurs (Whittle [132] par exemple), le lecteur intéressé est renvoyé à un article de Jennane *et al.* [70], qui propose une bonne et exhaustive revue sur l'estimation de H.

2.7.1 Méthode R/S

La méthode R/S [19] est très populaire pour estimer le paramètre de Hurst. C'est aussi la plus ancienne. On peut définir, étant donnée une série de n observations $\{X_r(n)\}_{n=1,..,N}$, une statistique notée $R(N)/S(N)$ définie comme suit :

$$\frac{R(N)}{S(N)} = \frac{\max\{0, W(1), \ldots, W(N)\} - \min\{0, W(1), \ldots, W(N)\}}{\sqrt{\hat{\sigma}^2}}$$

$$W(n) = \sum_{i=1}^{n} \left(X_r(i) - \hat{\mu}\right)$$

$\hat{\mu}$ et $\hat{\sigma}^2$ sont les estimateurs de moyenne et de variance standards [67] (voir section 1.4.2).

Si X_r est la réalisation d'un processus à longue mémoire, alors on peut montrer que :

$$\exists c \in \mathbb{R}, \quad \mathbb{E}\left(\frac{R(N)}{S(N)}\right) \underset{N\to\infty}{\sim} cN^H \tag{2.8}$$

Cette relation implique que :

$$\exists c \in \mathbb{R}, \quad \log\left(\mathbb{E}\left(\frac{R(N)}{S(N)}\right)\right) \underset{N\to\infty}{\sim} \log(c) + H\log(N)$$

En pratique, afin d'avoir une méthode d'estimation robuste, l'analyse est basée sur une approche graphique [99]. L'idée est de diviser l'ensemble d'observations en M parties disjointes et consécutives. On calcule ensuite la statistique $R(t_i,K)/S(t_i,K)$, qui correspond à la statistique $R(K)/S(K)$ appliquée aux K observations prises à partir de t_i, chaque t_i correspondant à un début de partie ($t_i = i(M/N)), \forall i, t_i < N$). On calcule cette statistique pour différentes valeurs de K comprises entre 1 et N, et on trace avec une échelle logarithmique tous les points

$\big(R(t_i,K)/S(t_i,K), \forall i\big)$. Comme on a divisé l'ensemble d'observations, on a plusieurs points pour chaque valeur de K et on peut faire maintenant une régression linéaire robuste pour estimer H. En pratique on fait la moyenne à K fixé, pour chaque point de départ t_i des parties de l'ensemble d'observations.

Cette méthode a une valeur historique, néanmoins ses performances statistiques sont faibles par rapport aux deux méthodes présentées ci-après.

2.7.2 Méthode Temps-Variance

Une autre méthode pour estimer le paramètre H est d'utiliser l'équation (2.5) [19]. Cette relation implique en effet que l'évolution du logarithme de la variance du processus agrégé d'ordre m en fonction de $log(m)$ soit une droite de coefficient directeur $\alpha = 2(H-1)$:

$$\exists c \in \mathbb{R}, \quad \log\big(\mathrm{Var}(X^{(m)})\big) = \log(\sigma^2) + (2-2H)log(m) \tag{2.9}$$

Il suffit donc de tracer $\log\big(\mathrm{Var}(X^{(m)})\big)$ en fonction de $log(m)$ et de faire la régression linéaire afin d'obtenir une approximation de H via le coefficient directeur de la droite obtenue.

Cette méthode possède l'avantage d'être simple. De plus, elle dispose de meilleures performances statistiques que la méthode R/S. Elle possède néanmoins une limitation car pour donner de bons résultats, il faut que l'ensemble d'observations soit très grand. Cela est principalement dû au fait que la caractéristique de longue mémoire a pour conséquence que les estimateurs de moyenne et de variance standards ont des performances moins bonnes (la loi des grands nombres sur laquelle ils sont basés converge beaucoup moins vite). Il faut donc plus de points pour avoir des performances correctes, ce qui n'est pas toujours possible. C'est pour cette raison que d'autres méthodes ont été développées, comme celle à base d'ondelettes présentée dans la section suivante.

2.7.3 Estimateur basé sur les ondelettes

L'objectif ici n'est pas de reprendre toute la théorie des ondelettes, mais plutôt de montrer ce que les ondelettes ont apporté dans le cadre de l'estimation du paramètre H. Cette méthode est décrite en détail dans [2, 1, 152]. L'idée de base est la même que pour la méthode temps-variance présentée dans la section précédente, mais au lieu d'utiliser le processus agrégé, qui correspond à une projection sur une famille de fonctions *boîtes* comme cela a été montré dans la section 1.2.6, on va utiliser une autre famille de fonctions, des *ondelettes*, et obtenir ainsi de meilleures performances statistiques.

La transformée en ondelettes (définie et étudiée depuis une vingtaine d'années seulement) permet d'obtenir une représentation temps-échelle (ou temps-fréquence) du signal déterministe ou de la réalisation d'un processus stochastique. Il s'agit, comme toute transformation, d'une projection d'un signal dans un espace fonctionnel (voir section 1.2.6). Il s'agit donc de projeter le signal sur une famille de fonctions à deux paramètres : le temps (a), et les échelles (b). Par analogie, la transformée de Fourier peut être vue comme une projection du signal sur une famille de fonctions à un paramètre f : les exponentielles complexes $\{e^{-2i\pi f}\}$. La famille de fonctions dans le cas de la transformée en ondelettes correspond à des versions translatées et dilatées d'une fonction de base appelée ondelette mère $\psi_{0,0}$:

$$\{\psi_{a,b}\}_{a,b\in\mathbb{R}}, \quad \psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi_{0,0}\left(\frac{t-b}{a}\right) \tag{2.10}$$

L'ondelette mère $\psi_{0,0}$ doit être localisée en temps autour du point t_0 (en général l'origine des temps), et localisée en fréquence autour d'une fréquence f_0 (transformée de Fourier localisée autour de f_0), l'ondelette $\psi_{a,b}$ sera ainsi localisée en temps autour du point $t_0 + b$, et localisée en fréquences autour du point $f_0 + a$. Les ondelettes sont donc des fonctions qui sont à la fois localisées en temps, et en fréquence. En pratique, aucune fonction ne peut être à la fois très localisée en temps *et* en fréquence (c'est le principe d'incertitude, qui vient du fait que le produit d'une fonction et de sa transformée de Fourier possède une borne inférieure), il faudra donc trouver un bon compromis, et assez peu de fonctions sont de bonnes ondelettes. Ces fonctions sont souvent comparées à un microscope mathématique, puisqu'elles permettent d'analyser le contenu fréquentiel situé autour de la fréquence a, au voisinage temporel de b.

Voici la définition de la transformée en ondelettes *continue* d'un signal $s(t), t \in \mathbb{R}$:

$$T(a,b) = \frac{1}{\sqrt{a}} < s(t), \psi_{a,b}(t) >= \frac{1}{\sqrt{a}} \int_{\mathbb{R}} s(t) \phi_{0,0}(\frac{t-b}{a}) dt \tag{2.11}$$

Pour que cela soit valable il faut que l'ondelette mère ait une moyenne nulle ($\int_{\mathbb{R}} \psi_{0,0}(t) dt = 0$), c'est d'ailleurs cette condition qui explique leur nom. En effet, pour que la moyenne soit nulle, il faut que la fonction oscille autour de 0. Étant donnée en plus qu'on va chercher des ondelettes localisées autour d'un point, cela implique de *petites oscillations*, donc des ondelettes. Il faut aussi, toujours pour que la transformée en ondelette soit valide, que la fonction à analyser ait une énergie finie ($\int_{\mathbb{R}} s(t)^2 dt < \infty$). Rappelons que l'ensemble des fonctions réelles d'énergie finie est noté $L^2(\mathbb{R})$.

Calculer la transformée en ondelettes continue d'un signal n'a pour nous pas de sens, car nous travaillons en temps discret. Il existe une version discrète de cette transformation, qui a de plus l'avantage de se calculer très rapidement. Soit une famille de fonctions dérivées d'une ondelette mère $\psi_{0,0}$:

$$\{\psi_{j,k}\}_{j,k \in \mathbb{N}}, \quad \psi_{j,k}(n) = 2^{\frac{-j}{2}} \psi_{0,0}(2^{-j}n - k) \tag{2.12}$$

On obtient les coefficients d'ondelette notés $d_X(j,k)$ d'une réalisation d'un processus stochastique $\{X[n]\}_{n \in \mathbb{N}}$ noté $X_r(n)$ en la projetant sur cette famille :

$$< X_r, \psi_{j,k} >= d_X(j,k) \tag{2.13}$$

$d_X(j,k)$ peut alors être considéré comme la réalisation d'un processus stochastique, dont les propriétés peuvent être reliées à celles du processus X. En particulier, si le processus X a pour spectre $\Gamma_X(\nu)$, et que l'ondelette mère $\psi_{0,0}(n)$ a pour spectre $\Psi_{0,0}(\nu)$ on peut montrer la relation suivante :

$$\mathbb{E}(|d_X(j,k)|^2) = \int_{\mathbb{R}} \Gamma_X(\nu) 2^j |\Psi_0(2^j \nu)|^2 d\nu \tag{2.14}$$

Remarque : $\mathbb{E}(|d_X(j,k)|^2)$ représente le moment d'ordre 2 du processus $\{d_X(j,k)\}_{k \in \mathbb{N}}$, à j fixé. C'est la moyenne du carré des coefficients d'ondelettes sur l'ensemble des k coefficients disponibles à l'échelle j.

La longue mémoire impliquant que le spectre diverge aux alentours de l'origine (voir section 2.1), on peut montrer que, si X possède une longue mémoire de paramètre H, alors on a [2] :

$$\forall j, \exists c > 0 \quad \mathbb{E}(|d_X(j,k)|^2) = c(2^j)^{2H} = c2^{(2jH)} \tag{2.15}$$

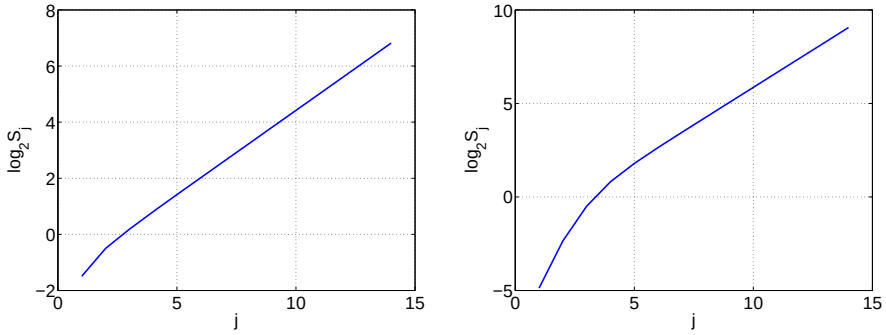


Fig. 2.3: Diagrammes LD d'un FGN de paramètre H=0.8 (a) et d'un FARIMA$(0.5,.4,-0.3)$ (b).

Introduisons maintenant la *fonction de structure*, qui correspond à l'estimation standard de l'espérance de $|d_X(j,k)|^2$:

$$S(j) = \frac{1}{n_j} \sum_{k=1}^{n_j} |d_X(j,k)|^2 \tag{2.16}$$

n_j étant le nombre de coefficients disponibles à l'échelle j.

En utilisant l'équation (2.15), on peut facilement obtenir :

$$\exists c > 0 log_2(S(j)) = log_2(c) + 2jH \tag{2.17}$$

Cette relation permet de comprendre comment le paramètre de longue mémoire H va pouvoir être estimé. En effet, l'évolution de $log_2(S(j))$ doit être, si le processus possède de la longue mémoire, une droite dont le coefficient directeur a est directement relié à H ($a = 2H$). En faisant la régression linéaire sur la gamme des grandes échelles, on obtient une bonne approximation de H.

Le tracé de $S(j)$ en fonction de 2^j peut être vu comme une représentation de la densité spectrale de puissance du processus, avec l'axe des abscisses inversé car les petites échelles correspondent aux variations rapides donc aux hautes fréquences. Nous utiliserons beaucoup dans ce texte cette représentation de la covariance, que l'on trace dans un diagramme log-log ($\log_2(S(j))$ en fonction de $\log_2(2^j) = j$). On le dénommera diagramme LD (pour *Logscale Diagram*) par la suite. Le diagramme LD théorique d'un FGN ($H = 0.8$) et d'un FARIMA$(0.5,.4,-0.3)$ est présenté sur la figure 2.3.

Remarque : l'ondelette choisie pour la décomposition $\psi_{0,0}$ n'est pas censée influencer les résultats, cependant une de ses caractéristiques est importante : son nombre de moments nuls M qui indique que l'ondelette sera orthogonale à tous les monômes de degré inférieur ou égale à M :

$$M = \max_{m \in \mathbb{N}} \{ < \psi_{j,k}, 1 >= 0, < \psi_{j,k}, n^1 >= 0, \ \dots \ , < \psi_{j,k}, n^m >= 0 \} \tag{2.18}$$

Ceci est très utile si on considère un processus comme étant la somme d'une fonction f polynomiale de degré M et d'un processus stochastique à longue mémoire X. On module alors la moyenne du processus X par une fonction polynomiale, et cela perturbe de manière significative les estimations des propriétés statistiques du processus de fluctuations autour de cette moyenne. Comme on s'intéresse ici à l'estimation du paramètre de longue mémoire, on ne veut pas tenir compte de cette variation. Or, étant donné que la transformée en ondelette est une transforma-

tion linéaire par linéarité du produit scalaire de $L^2(\mathbb{N})$, on a :

$$< \psi_{j,k}, X_r + f >=< \psi_{j,k}, X_r > + < \psi_{j,k}, f >$$

Si l'ondelette a M moments nuls, on a alors :

$$< \psi_{j,k}, X_r + f >=< \psi_{j,k}, X >$$

La transformée en ondelettes ne tient donc pas compte des variations polynomiales de degré au plus M qui pourraient être superposées à la réalisation d'un processus stochastique que l'on souhaite étudier.

Pourquoi utiliser un estimateur basé sur les ondelettes ?

- On peut montrer que les processus $d_X(j,k)$ possèdent une courte mémoire (décroissance exponentielle de la covariance) si $M > H + 1/2$ [2], ce qui permet d'utiliser la loi des grands nombres dans de meilleures conditions que sur le processus original. La performance statistique de l'estimation est donc meilleure.
- Sous l'hypothèse que le processus est gaussien et que les variables aléatoires $d_X(j,k)$ sont indépendantes (ce qui est justifié par la courte mémoire), alors les intervalles de confiances peuvent être analytiquement calculés. En pratique on observe un biais négligeable proche du MLE (*Maximum Likehood Estimation*, voir section 1.4.2) [2].
- En contrôlant M, le nombre de moments nuls de l'ondelette, on peut supprimer certaines tendances inintéressantes dans les données comme cela a été dit plus haut.

Cet estimateur a été implémenté dans MATLAB par Patrice Abry et Darryl Veitch et ces codes sont disponibles en ligne (http://perso.ens-lyon.fr/patrice.abry/).

2.8 Synthèse de processus LRD gaussiens

La génération de réalisations de processus à longue mémoire est un sujet qui a reçu beaucoup d'attention [19, 13, 39, 59]. Il est en effet intéressant de disposer de méthodes pour produire des réalisations de tels processus pour d'une part tester la performance statistique de l'estimation du paramètre de Hurst, et d'autre part utiliser ces réalisations synthétiques pour effectuer de la prédiction ou remplacer un ensemble d'observation. Il existe maintenant des algorithmes bien établis pour synthétiser des réalisations de processus gaussiens ayant une fonction de covariance donnée $\gamma(i,j)$ [13]. Nous allons dans cette section faire une revue des principaux algorithmes permettant d'obtenir une réalisation de N échantillons d'un processus X, gaussien de fonction de covariance $\gamma(k)$.

Étant donné que, afin de générer des réalisations de processus non gaussiens à longue mémoire, on se base systématiquement sur des processus gaussiens, ces algorithmes nous seront utiles par la suite. Les méthodes décrites ci-après sont toutes exactes, c'est à dire que l'on peut montrer que la covariance voulue est bien présente dans la réalisation synthétique. C'est aussi pourquoi ces différentes méthodes sont uniquement comparées du point de vue de la rapidité de génération.

2.8.1 Décomposition de Cholesky

L'idée de base de cette méthode vient de la propriété fondamentale suivante. Soit $\Sigma_{i,j} = \gamma(i,j)$ la matrice de covariance (la fonction de covariance se réduit à une fonction d'une seule variable

dans le cas de processus stationnaire au sens large, voir section 1.2.4). On peut calculer *la racine carrée* de cette matrice, c'est à dire la matrice A, telle que sa transposée A^t multipliée par elle-même soit égale à $\Sigma = A^t A$. On peut montrer que pour obtenir une réalisation d'un processus gaussien X ayant pour matrice de covariance Σ, il faut prendre $X_r = A\epsilon$, ϵ étant la réalisation d'un processus IID gaussien de moyenne nulle. Comme nous nous intéressons à la synthèse de processus stationnaire, et que donc $\gamma(i,j) = \gamma(0, j-i) = \gamma(k)$, la matrice Σ a la forme suivante :

$$\begin{pmatrix} \gamma(0) & \gamma(1) & \cdots & \gamma(N) \\ \gamma(1) & \gamma(0) & \cdots & \gamma(N-1) \\ & \cdots & & \\ \gamma(N) & \gamma(N-1) & \cdots & \gamma(0) \end{pmatrix}$$

Pour calculer A, on peut utiliser la décomposition de Cholesky [110], qui permet d'obtenir une matrice A triangulaire. Ceci n'est valable que si ϵ est une réalisation d'un processus IID gaussien, c'est pourquoi on ne peut obtenir par cette technique que des réalisations de processus à longue mémoire gaussiens.

La complexité de cet algorithme est $O(N^3)$, elle est donc importante, et nous allons voir dans les sections suivantes que l'on peut obtenir le même résultat plus rapidement.

2.8.2 Méthode de la matrice circulante

Cette méthode consiste à construire une matrice circulante carrée de taille $M \geq 2(N-1)$ à partir de la covariance $\gamma(k), k \in 0..N$ [39]. Les matrices circulantes disposent d'une propriété attrayante : elles peuvent être diagonalisées en utilisant l'algorithme de transformée de Fourier rapide. Rappelons qu'une matrice est circulante si elle est de la forme suivante :

$$\begin{pmatrix} x_0 & x_1 & \cdots & x_n \\ x_n & x_0 & \cdots & x_{n-1} \\ & \cdots & & \\ x_1 & x_2 & \cdots & x_0 \end{pmatrix}$$

Concrètement, la procédure de synthèse est la suivante [13] :

- On choisit $M = 2^p \geq 2(N-1)$
- On définit ensuite le vecteur $V = (\gamma(0), \gamma(1), \ldots, \gamma(M/2-1), \gamma(M/2), \ldots, \gamma(1))$
- On calcule W, la transformée de Fourier de V. Ici il faut s'assurer que W possède des valeurs positives ou nulle, dans le cas contraire, il faut choisir p plus grand.
- On génère indépendamment deux réalisations indépendantes de processus gaussiens IID de moyenne nulle et de variance $1/\sqrt{2}$, notées X et Y.
- On introduit $Z = X + iY$.
- On définit enfin le vecteur U tel que $U(k) = \sqrt{W(k)}Z(k)$
- On obtient en prenant la transformée de Fourier inverse de U deux réalisations indépendantes (partie réelle et partie imaginaire) d'un processus gaussien, de moyenne nulle et de variance 1, ayant la covariance $\gamma(k)$.

Cette méthode est très rapide ($O(Nlog(N))$) grâce à l'algorithme de transformée de Fourier rapide, mais elle ne permet pas de faire une génération *en ligne* du processus car l'ensemble des

N valeurs est généré d'un seul coup. Pour cela, il faut utiliser la récursion de Durbin-Levinson présentée dans la section suivante.

2.8.3 Récursion de Durbin-Levinson

L'algorithme de Durbin-Levinson [25, 13] permet d'exprimer la réalisation d'un processus X gaussien de covariance $\gamma(k)$ de manière récursive :

$$X[n+1] = \Phi_{n,1}X[n] + \ldots + \Phi_{n,n}X[1] + \sqrt{v_n}\epsilon[n+1]$$

$$X[1] = \gamma(0)\epsilon[1]$$

ϵ est un processus IID gaussien de moyenne nulle et de variance 1. Il est possible, et c'est d'ailleurs le cœur de l'algorithme, de calculer récursivement les coefficients $\Phi_{i,j}$ et les variances v_n par le biais de la définition suivante, qui provient des équations de Yule-Walker [25] :

$$\begin{aligned}
\Phi_{n,n} &= \frac{1}{v_{n-1}}\left[\gamma(n) - \sum_{j=1}^{n-1}\Phi_{n-1,j}\gamma(n-j)\right] \\
\Phi_{n,i} &= \Phi_{n-1,i} - \Phi_{n,n}\Phi_{n-1,n-i}, \quad i < n \\
v_n &= v_{n-1}(1-\Phi_{n,n}^2) \\
v_0 &= \gamma(0)
\end{aligned}$$

On peut ainsi construire un algorithme pour générer en ligne un processus LRD gaussien, et sa complexité est $O(N^2)$ pour le calcul du $N^{\text{ième}}$coefficient. Nous avons implémenté cet algorithme, mais en pratique il est trop lent pour être utilisé. En effet la complexité pour générer une réalisation de taille N est en $O(N^3)$ car on a :

$$\sum_{k=1}^{N} k^2 = \frac{1}{6}N(N+1)(2N+1) \underset{N\to+\infty}{\sim} O(N^3)$$

2.8.4 Autres méthodes

D'autres méthodes comme la méthode spectrale (on cherche à générer le spectre du processus pour en prendre la transformée de Fourier inverse), ainsi que d'autres méthodes basées sur les matrices sont détaillés dans [13].

Nous avons toujours utilisé la méthode de la matrice circulante, qui reste à l'heure actuelle la méthode exacte la plus rapide pour obtenir des réalisations de processus gaussiens de covariance prescrite.

2.9 Au-delà de l'autosimilarité

Des modèles plus complexes que ceux incluant de l'autosimilarité ou de la longue mémoire ont été proposés pour modéliser le trafic Internet. Dans un processus autosimilaire, le paramètre H est constant, il ne dépend pas du temps. Il existe un formalisme qui considère des variations de ce paramètre H, il s'agit du formalisme *multifractal* [48, 37], qui fait intervenir des statistiques d'ordres supérieurs.

Nous avons pas envisagé les processus multifractals dans notre étude du trafic Internet et du trafic sur puce. La synthèse de processus multifractals est en effet un problème complexe, dans lequel on perd souvent le contrôle précis sur la loi marginale, et nous avons préféré focaliser nos travaux sur la modélisation précise des statistiques d'ordre un et deux. De plus, le fait que les les données de trafic Internet aient un comportement multifractal est encore à l'heure actuelle un sujet de débat [144, 61].

2.10 Récapitulatifs des processus

Nous proposons ici différents tableaux récapitulatifs sur les processus qui ont été introduits dans ce texte et qui seront utilisés par la suite. Le tableau 2.1 (page 44) expose différentes lois marginales classiques, et le tableau 2.2 (page 45) répertorie quelques densités spectrales de puissance (représentation fréquentielle de la covariance). La figure 2.4 (page 44)montre des exemples de fonctions de masse (PDF) pour quelques lois classiques, et la figure 2.5 (page 44) montre des exemples de densité spectrales de puissance. On peut noter la signature de la longue mémoire dans la densité spectrale de puissance par le comportement dans les basses fréquences qui est en loi de puissance, ce qui se matérialise par une droite dans un diagramme log-log. Dans le cas IID et ARMA, qui sont des processus à courte mémoire, la droite a pour coefficient directeur 0 ($H = 0.5$). La figure 2.5 montre aussi bien la correspondance qui peut être faite entre un processus à longue mémoire (FGN), un processus à courte mémoire (ARMA) et un processus combinant des comportements à courte et à longue mémoire (FARIMA).

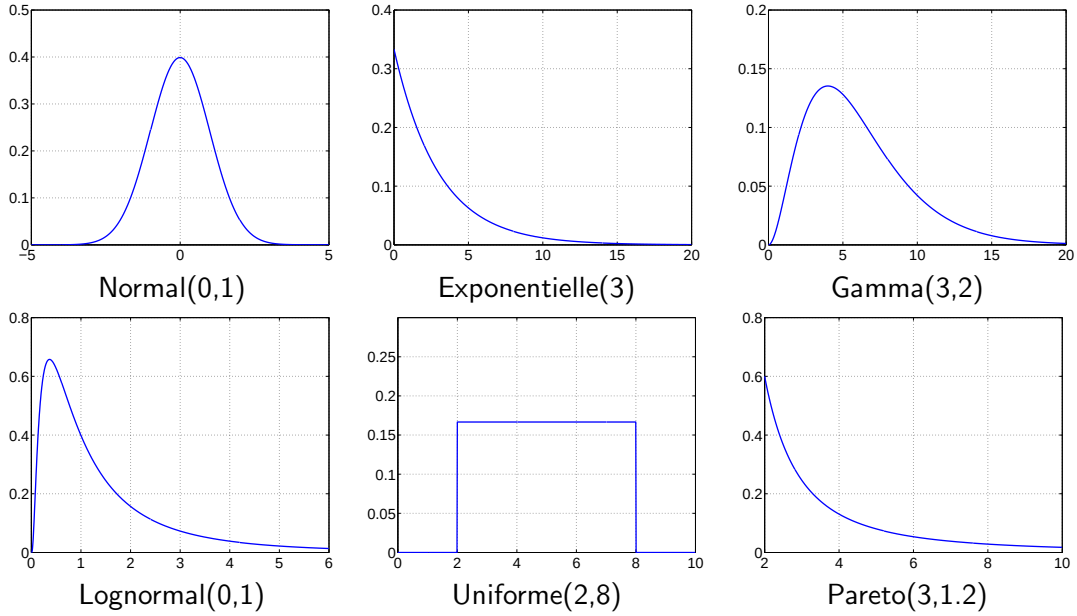


Fig. 2.4: Quelques densités de probabilité (PDF) continues.

Loi marginale	Densité de probabilité	Paramètres	Moyenne	Variance
Uniforme	$\frac{1}{b-a}$	a, b	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
Exponentielle	$\lambda e^{-\lambda x}$	λ	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
Normale	$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	μ, σ^2	μ	σ^2
Gamma	$\frac{\beta^\alpha x^{\alpha-1} e^{-\beta x}}{\Gamma_f(a)}$	α, β	$\alpha\beta$	$\alpha\beta^2$
LogNormal	$\frac{1}{(x-\mu)\sigma\sqrt{2\pi}} e^{-\frac{\ln((x-\mu))^2}{2\sigma^2}}$	μ, σ^2	$e^{\sigma^2/2}$	$e^{2\mu+\sigma^2}(e^{\sigma^2}-1)$
Pareto	$\alpha\frac{\beta^\alpha}{x}$	α, β	$\frac{\alpha\beta^2}{\alpha-1}$	$\frac{\alpha\beta^2}{(\alpha-1)^2(\alpha-2)}$

Tab. 2.1: Quelques lois continues classiques.

Propriété	$\Gamma(\nu)$	Commentaire
Longue mémoire	$\sim \nu^{-\alpha}, \nu \to 0$	$0 \leq \alpha \leq 1$
Processus $1/f$	$\sim \nu^{-\alpha}, \nu_1 \leq \nu \leq \nu_2$	$0 \leq \alpha \leq 1$, autosimilarité
Processus fractals	$\sim \nu^{-\alpha}, \nu \to +\infty$	$0 \leq \alpha \leq 1$
ARMA	$\frac{\sigma^2}{2\pi} \frac{\lvert\Phi_p(e^{-2i\pi\nu})\rvert^2}{\lvert\Theta_q(e^{2i\pi\nu})\rvert^2}$	Courte mémoire
FARIMA	$\frac{\sigma^2}{2\pi} \frac{\lvert\Phi_p(e^{-2i\pi\nu})\rvert^2}{\lvert\Theta_q(e^{2i\pi\nu})\rvert^2} \lvert(1-e^{2i\pi\nu})\rvert^{2d}$	$0 \leq d \leq 1/2$ Autosimilarité $(H = d + 1/2)$
Bruit blanc	$\sim C$	Processus IID
Bruit Rose	$\sim \nu^{-1}$	Autosimilarité $(H = 0)$
Bruit Marron	$\sim \nu^{-2}$	Autosimilarité $(H = 1)$
Bruit Noir	$\sim \nu^{-(2H+1)}$	Autosimilarité $(H > 1/2)$

Tab. 2.2: Quelques densités spectrales de puissance.

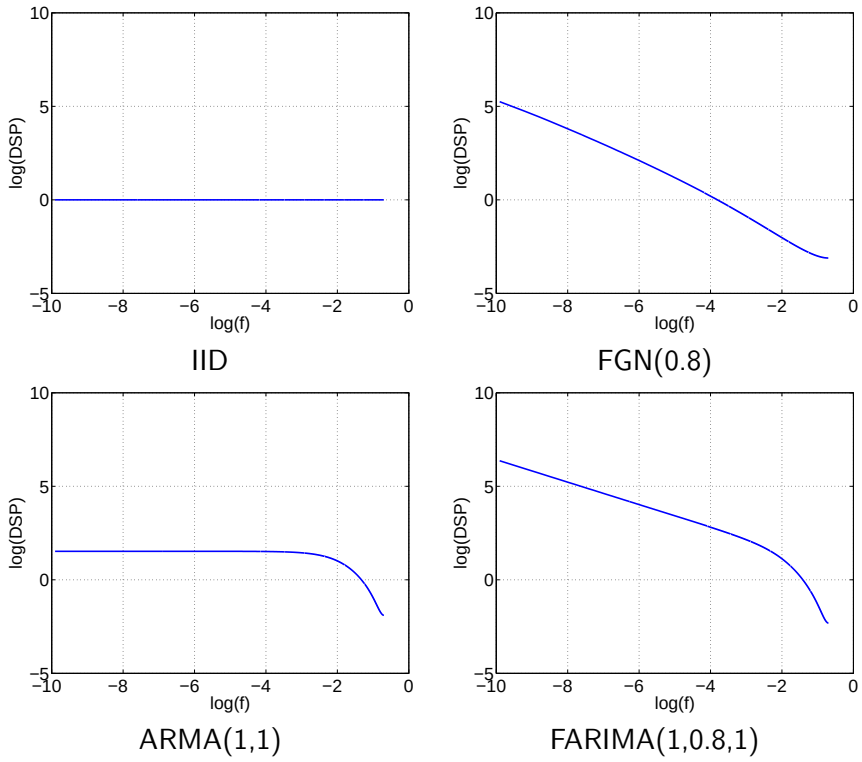


Fig. 2.5: Quelques densités spectrales de puissance tracées dans une échelle log-log (diagrammes LD).

2.11 Conclusion

Nous avons dans ce chapitre introduit les différents modèles de processus stochastiques que nous allons utiliser dans les chapitres suivants dans le cadre de la modélisation et la synthèse de trafic Internet ainsi que dans le cadre de l'analyse et la synthèse de trafic sur puce. Il existe de nombreux autres modèles de processus stochastiques, nous nous sommes restreints dans cette étude aux processus stationnaires, et à la modélisation des statistiques d'ordre un (loi marginale) et deux (covariance). Cette modélisation conjointe permet de couvrir une large gamme de situations et nous avons considéré que cela était suffisant pour les modélisations que nous avons effectuées. La prise en compte conjointe de la loi marginale et de la covariance implique de pouvoir générer des réalisations de processus ayant une forme de covariance et une forme de loi marginale données. Cette problématique est discutée dans le chapitre suivant.

Chapitre 3
Synthèse de processus LRD non-gaussiens

Ce chapitre présente la problématique de la synthèse de réalisations de processus à longue mémoire non-gaussiens. Cette étude a été réalisée d'une part car les premières données de trafic sur puce que nous avons analysées contenaient des signes de longue mémoire et que leurs lois marginales étaient très éloignées d'une loi Normale, d'autre part car la plupart des traces de trafic Internet que nous avons analysées ont la même propriété. C'est pourquoi nous avons recherché des méthodes pour générer des processus à longue mémoire suivant une loi donnée différente de la loi Normale. Il est vite apparu que, si la génération de réalisations de processus gaussiens de tout type était un problème déjà bien exploré comme cela a été montré dans le chapitre précédent, le cas non-gaussien en revanche ne semblait pas avoir reçu beaucoup d'attention. Ce travail nous a permis de mettre au point une méthode de synthèse de réalisations de tels processus. Notre approche est décrite en détail à la section 3.1 et des résultats expérimentaux sont discutés dans la section 3.2.

Cette collaboration a été le point de départ des travaux présentés dans le chapitre 4 et a fait l'objet d'une publication de synthèse dans le journal TDSC (*IEEE Transaction on Dependable and Secure Computing*) [SLB+07] ainsi que de publications dans des conférences internationales [SLB$^+$06a] et francophones [SA05, SLB$^+$06b].

3.1 Contribution à l'étude du cas non-gaussien

Les algorithmes présentés dans le chapitre précédent permettent de générer des réalisations de processus à longue mémoire dont la loi marginale est gaussienne. Il est en pratique difficile d'arriver à générer un processus ayant une loi marginale donnée *et* une covariance donnée. Il faut pour cela partir d'un processus gaussien et le modifier pour obtenir ce que l'on souhaite. Après la revue d'une méthode existante dans la section 3.1.1, notre approche sera détaillée dans la section 3.1.2.

3.1.1 Méthode de l'inverse

On cherche, étant donné un processus gaussien X (ayant une CDF gaussienne notée F_X et une covariance $\gamma_X(k)$), un processus Y ayant une CDF notée F_Y et une covariance γ_Y données. On a, par définition de la loi Normale :

$$F_X(x) = \frac{1}{2}\left[\mathrm{erf}(\frac{x}{\sqrt{2}}) + 1\right]$$

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

Une méthode intuitive pour transformer un processus gaussien en un processus non-gaussien est de prendre une fonction h du processus X, on change ainsi sa loi marginale et on peut la faire correspondre avec loi recherchée F_Y en appliquant :

$$\forall n \in \mathbb{N}, \quad Y[n] = h(X[n]) = F_Y^{-1}(F_X(X[n]))$$

En effet, en appliquant la fonction $F_X(X[n])$, les variables aléatoires $X[n]$ ayant pour CDF F_X, on obtient des variables aléatoires suivant une loi uniforme [55]. A partir d'une variable aléatoire uniforme on peut obtenir n'importe quelle distribution en appliquant la fonction de répartition (CDF) inverse [55]. On peut donc obtenir Y à partir de X avec la fonction $h = F_Y^{-1} o F_X$. Dans cette méthode, on ne change pas la covariance : $\gamma_X = \gamma_Y$.

Cette technique est très attractive car elle est très simple est très rapide. Il suffit de générer un processus X de covariance γ_X à l'aide de l'une des méthodes décrites à la section 2.8 et d'appliquer la fonction h au processus X pour obtenir le processus Y voulu. Néanmoins, lorsque l'on applique la fonction h d'un processus, la covariance est aussi modifiée. Huang *et al.* a montré dans l'article [62], que si la fonction h est de carré intégrable ($\int h^2(t)dt < \infty$) et que $\mathbb{E}(h(X)X) \neq 0$, alors la forme asymptotique de la covariance est préservée, c'est à dire que si X possède une longue mémoire de paramètre H, alors Y aussi. La méthode n'est donc pas exacte, puisque seul un comportement asymptotique est garanti avec des restrictions sur la fonction h donc sur la forme de la marginale que l'on souhaite obtenir.

3.1.2 Notre approche

L'approche que nous avons développée est de partir, comme pour la méthode de l'inverse, d'un processus X gaussien ayant une covariance γ_X, puis de prendre une fonction de X afin d'obtenir un processus $Y = f(X)$ ayant les caractéristiques que l'on souhaite, c'est à dire une loi marginale et une covariance données. Nous nous sommes de plus fixés comme objectif de reproduire de manière exacte la covariance, c'est pourquoi nous allons chercher à trouver comment fixer la covariance de X de telle sorte que l'on obtienne exactement la covariance voulue pour Y.

Le cœur du problème consiste donc à trouver $\gamma_X(k)$ connaissant $\gamma_Y(k)$, autrement dit avec quelle covariance synthétiser le processus gaussien, pour que quand on applique la fonction f, il ait exactement la covariance souhaitée. On s'attend à obtenir une relation $\gamma_Y(k) = F(\gamma_X(k))$. L'objectif est donc connaissant f, de trouver F ou plutôt son inverse F^{-1}. Il suffit alors de générer un processus gaussien ayant une covariance $\gamma_X(k) = F^{-1}(\gamma_Y(k))$. On applique ensuite la fonction f et on obtient le processus Y recherché, ayant la loi marginale et la covariance désirées.

Ce problème a été étudié par Teich et Lowen dans le cadre d'une application de neurosciences [94] et nous avons utilisé quelques résultats établis par eux pour construire notre méthode de synthèse. Nous avons aussi découvert, après avoir largement commencer à explorer une méthode, que Crouse et Baraniouk avait utilisé une approche similaire pour générer des réalisations de processus à longue mémoire non-gaussiens [34]. L'article ne semble pas avoir été publié finalement, mais la version soumise est accessible sur le WEB et elle est parfois citée dans des publications. Nos calculs sont en accord avec les leurs sur les lois marginales qu'ils ont considérées (uniforme, exponentielle, lognormale et Pareto), nous avons obtenu des résultats similaires pour les lois Gamma et χ^2.

Notre approche a été publiée en français au *GRETSI 2005* [SA05], elle est aussi détaillée dans le rapport de recherche [SLO+05] qui est la base d'une communication dans le journal *IEEE TDSC* [SLB+07].

3.1.2.1 Calculs pour une loi du χ^2

Voici la résolution du problème pour le cas d'une loi marginale du χ^2 à m degrés de liberté.

- Soit $\{X_i\}_{i=1...m}$ une famille de processus indépendants gaussiens de moyenne nulle et de variance $\sigma_X^2 = 1$ ayant tous la même covariance $\gamma_{X_i}(k) \triangleq \gamma_X(k)$.
- Soit Y un processus suivant une loi du χ^2 à m degrés de liberté et ayant pour covariance $\gamma_Y(k)$. C'est le processus que l'on souhaite obtenir.
- On a par définition d'une loi du χ^2 [44] :

$$\forall n, \quad Y[n] = \sum_{i=1}^{m} X_i^2[n] \tag{3.1}$$

- On peut montrer que $\forall n, \quad \mathbb{E}(Y[n]) = m$
- Notre objectif est de trouver une relation entre γ_X et γ_Y.

Par définition de la covariance (voir section 1.2.3), on a :

$$\begin{aligned}\gamma_Y(k) &= \mathbb{E}\big(Y[n]Y[n+k]\big) - \mathbb{E}\big(Y[n]\big)^2 \\ &= \mathbb{E}\big(\sum_{i=1}^{m} X_i^2[n] \sum_{j=1}^{m} X_j^2[n+k]\big) - m^2 \\ &= \sum_{i=1}^{m}\sum_{j=1}^{m} \mathbb{E}\Big(X_i^2[n]X_j^2[n+k]\Big) - m^2\end{aligned}$$

Par définition, les variables aléatoires $X_i[n]$ et $X_j[n]$ sont indépendantes pour tout n et pour tout couple (i, j) tel que $i \neq j$. En utilisant le fait que l'espérance du produit de deux variables aléatoires indépendantes est égale au produit des espérances (voir section 1.1.4), on a :

$$\begin{aligned}\gamma_Y(k) &= \sum_{i=1}^{m}\Big[\mathbb{E}\big(X_i^2[n]X_i^2[n+k]\big) + \sum_{j=1,j\neq i}^{m} \mathbb{E}\big(X_i^2[n]X_j^2[n+k]\big)\Big] - m^2 \\ &= \sum_{i=1}^{m}\Big[\mathbb{E}\big(X_i^2[n]X_i^2[n+k]\big) + \sum_{j=1,j\neq i}^{m} \mathbb{E}\big(X_i^2[n]\big)\mathbb{E}\big(X_j^2[n+k]\big)\Big] - m^2 \\ &= \sum_{i=1}^{m}\Big[\mathbb{E}\big(X_i^2[n]X_i^2[n+k]\big) + \sum_{j=1}^{m-1} 1\Big] - m^2\end{aligned} \tag{3.2}$$

Il reste à calculer $\mathbb{E}\big(X_i^2[n]X_i^2[n+k]\big)$, et $X_i[n]$ et $X_i[n+k]$ ne sont pas indépendantes. Pour cela, nous avons utiliser une technique décrite dans [94], qui permet de décomposer $X_i[n+k]$ en deux parties telles que :

- Une soit proportionnelle $X_i[n]$
- Une soit décorrélée de $X_i[n]$

Le calcul étant identique pour tout i, nous utiliserons X pour se référer à un processus X_i afin de simplifier les notations. On pose donc la décomposition suivante :

$$X[n+k] = \rho_X(k)X[n] + Z[n,k], \qquad \rho_X(k) = \frac{\gamma_X(k)}{\sigma_X^2} = \gamma_X(k) \tag{3.3}$$

On peut montrer que les variables aléatoire $Z[n,k]$ sont gaussiennes et que leurs moyennes sont nulles. On a de plus :

$$\begin{aligned}\gamma_X(k) &= \mathbb{E}\big((X[n]X[n+k]\big)\\ &= \mathbb{E}\big((X[n](\rho_X(k)X[n]+Z[n,k])\big)\\ &= \rho_X(k)\mathbb{E}\big(X^2[n]\big)+\mathbb{E}\big(X[n]Z[n,k]\big)\\ \gamma_X(k) &= \gamma_X(k)+\mathbb{E}\big(X[n]Z[n,k]\big)\end{aligned}$$

Ainsi $\mathbb{E}\big(X[n]Z[n,k]\big)=0$ et comme $Z[n,k]$ et $X[n]$ sont des variables aléatoires gaussiennes, on peut montrer [94] que $\{Z[n]\}_{n\in\mathbb{N}}$ et $\{X[n]\}_{n\in\mathbb{N}}$ sont indépendantes.

On peut calculer la variance de $Z[n]$:

$$\begin{aligned}\mathbb{E}\big(Z[n,k]^2\big) &= \mathbb{E}\Big[\Big[X[n+k]-\gamma_X(k)X[n]\Big]^2\Big]\\ &= \mathbb{E}\big(X^2[n+k]\big)+\gamma_X^2(k)\mathbb{E}\big(X^2[n]\big)-2\gamma_X^2(k)\mathbb{E}\big(X^2[n]\big)\\ &= 1+\gamma_X^2(k)-2\gamma_X^2(k)\\ &= 1-\gamma_X^2(k)\end{aligned} \tag{3.4}$$

En utilisant ces résultats, on peut calculer $\mathbb{E}\big(X_i^2[n]X_i^2[n+k]\big)$:

$$\begin{aligned}\mathbb{E}\big(X_i^2[n]X_i^2[n+k]\big) &= \mathbb{E}\Big[X^2[n]\Big[\gamma_X^2(k)X^2[n]+Z[n,k]^2+2\gamma_X(k)X[n]Z[n,k]\Big]\Big]\\ &= \gamma_X^2(k)\mathbb{E}\big(X[n]^4\big)+\mathbb{E}\big(X^2[n]\big)\mathbb{E}\big(Z[n,k]^2\big)+2\gamma_X(k)\mathbb{E}\big(X[n]^3\big)\mathbb{E}\big(Z[n,k]\big)\end{aligned}$$

On a de plus $\mathbb{E}\big(X[n]^3\big)=0$ et $\mathbb{E}\big(X[n]^4\big)=3$ car $X[n]$ est gaussien.

$$\begin{aligned}\mathbb{E}\big(X_i^2[n]X_i^2[n+k]\big) &= 3\gamma_X^2(k)+1-\gamma_X^2(k)\Big)\\ &= 2\gamma_X^2(k)+1\end{aligned}$$

En utilisant ce résultat dans l'équation (3.2), on obtient :

$$\begin{aligned}\gamma_Y(k) &= \sum_{i=1}^{m}\Big(2\gamma_X^2(k)+1+\sum_{j=1}^{m-1}1\Big)-m^2\\ \gamma_Y(k) &= 2m\gamma_X^2(k)+m^2-m^2\end{aligned}$$

$$\boxed{\gamma_Y(k)=2m\gamma_X^2(k)} \tag{3.5}$$

En inversant ce résultat, on a :

$$\boxed{\gamma_X(k)=\sqrt{\frac{\gamma_Y(k)}{2m}}} \tag{3.6}$$

On a donc une relation entre γ_X et γ_Y. Pour obtenir un processus Y suivant une loi du χ^2 à m degrés de liberté, il faut générer m processus gaussiens indépendants de moyenne nulle et de variance $\sigma_X^2=1$, ayant tous une covariance que l'on peut calculer en fonction de la covariance de

Loi	Paramètres	Définition	Relation
Normal	μ, σ	$Y[n] = X[n]$	$\gamma_X(k) = \gamma_Y(k)$
Chi-deux	m	$Y[n] = \sum_{i=1}^{m} X_i^2[n]$	$\gamma_X(k) = \sqrt{\frac{\gamma_Y(k)}{2m}}$
Exponentielle	μ	$Y[n] = \frac{\mu}{2}\left(X_1^2[n] + X_2^2[n]\right)$	$\gamma_X(k) = \frac{\sqrt{\gamma_Y(k)}}{\mu}$
Gamma	α, β	$Y[n] = \sum_{i=1}^{\alpha} \frac{\beta}{2}\left(X_{1,i}^2[n] + X_{2,i}^2[n]\right)$	$\gamma_X(k) = \sqrt{\frac{\gamma_Y(k)}{\alpha\beta^2}}$
Lognormale	μ, σ	$Y[n] = e^{X[n]}$	$\gamma_X(k) = \log\left(1 + \frac{\gamma_Y(k)}{\mu_Y^2}\right)$
Pareto	a, b	$Y[n] = be^{\frac{X_1^2[n]+X_2^2[n]}{2a}}$	$\gamma_X(k) = \frac{(a-1)^2\sqrt{\gamma_X(k)}}{a^2b^2 + \gamma_X(k)(a-1)^2}$
Uniforme	a, b	$Y[n] = a + (b-a)e^{-\frac{1}{2}(X_1^2[n]+X_2^2[n])}$	$\gamma_X(k) = 4\sqrt{\frac{\gamma_Y(k)}{(b-a)^2 + 4\gamma_Y(k)}}$

Tab. 3.1: Pour différentes lois, expression en fonction de processus gaussiens X_i, et relation entre γ_X et γ_Y garantissant une reproduction exacte de la covariance du processus Y.

recherché pour Y grâce à l'équation (3.6). Ainsi si un ensemble de donnée (du trafic Internet par exemple) que l'on cherche à modéliser possède, en plus d'une caractéristique de longue mémoire de paramètre H, une loi marginale bien approximée par une loi du χ^2, alors nous serons ainsi en mesure de générer du trafic synthétique dont les caractéristiques statistiques du premier et second ordre seront ajustées sur une trace.

Cette méthode est exacte, c'est à dire que la covariance de Y est entièrement reproduite contrairement à la méthode de l'inverse qui ne garantie qu'un comportement asymptotique.

Remarque : Ce résultat aurait pu être obtenu plus simplement en utilisant la résultat suivant : Soit 4 variables aléatoire gaussiennes indépendantes X_i, X_j, X_k et X_l, on a :

$$\mathbb{E}(X_i X_j X_k X_l) = \mathbb{E}(X_i X_j)E(X_k X_l) + \mathbb{E}(X_i X_k)E(X_j X_l) + \mathbb{E}(X_i X_l)E(X_j X_k)$$

En reportant ce résultat dans l'équation 3.2, on peut arrvier au résultat final (3.5). Néanmois, les techniques utilisés ici sont utile à la résolution du calcul pour d'autres lois marginales (voir annexe A).

3.1.2.2 Résultats des calculs pour d'autres lois

Nous avons fait les calculs lorsque $\{Y[n]\}_{n\in\mathbb{N}}$ suit les différentes lois marginales, le détail de ces calculs est reporté dans l'annexe A. Le tableau 3.1 contient d'une part la définition permettant d'obtenir un processus suivant chacune des lois marginales à partir d'une famille de processus gaussiens indépendants de moyenne nulle et de variance σ^2 [44] (colonne "Définition") et d'autre part la relation entre γ_Y permettant de fixer γ_X afin d'obtenir une reproduction exacte de la covariance γ_Y (colonne "Relation").

Remarque : Les calculs que nous avons mené aurait pu être fait à l'aide du théorème de Price [129], une note à ce sujet est inclus dans l'annexe A.

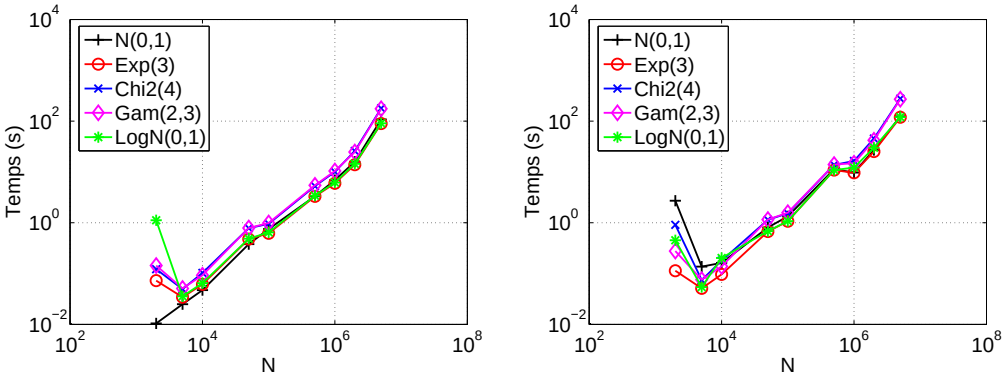


Fig. 3.1: Temps de génération en fonction de la taille N de la réalisation, pour un processus ayant la covariance d'un FGN (gauche) et d'un FARIMA (droite), et une marginale Normale (Nor), Exponentielle (Exp), χ^2, Gamma (Gam) et logNormale (logN). Les abscisses et les ordonnées sont en échelle logarithmique.

3.1.2.3 Restrictions

Les processus gaussiens sont générés à l'aide de la méthode de la matrice circulante (voir section 2.8.2). La complexité de la génération d'une réalisation de taille N est donc $O(Nlog(N))$.

De par la nature de cette méthode de synthèse, quelques restrictions sont à considérer :

- **Forme de la covariance** : Étant donné que l'on prend dans de nombreux cas la racine carrée de la covariance recherchée pour obtenir celle des processus gaussiens, cela impose dans ce cas que la covariance visée soit positive ou nulle. Dans le cas d'une covariance d'un FGN il n'y a aucun problème car celle-ci est positive (équation 2.4). Pour la covariance d'un FARIMA(ϕ,d,θ), cela impose que $\phi > \theta$.
- **Paramètres de la loi Gamma** : La définition d'un processus suivant une loi Gamma à partir de processus gaussiens implique que le paramètre de forme α soit un nombre entier. Nous sommes donc dans l'obligation d'arrondir α. La même restriction est à considérer pour le paramètre m dans le cas de la loi du χ^2.

Nous avons implémenté ces méthodes de synthèse sous la forme de scripts MATLAB ainsi que sous la forme d'une bibliothèque C++ sous licence GPL (GNU[1] *Public License*), utilisant et étendant la bibliothèque `newran`, développée par Robert Davis [36]. La bibliothèque inclut aussi des fonctions pour calculer la covariance d'un FARIMA et d'un FGN.

3.1.3 Temps de génération

Dans cette section, nous présentons des résultats de temps de génération. Cette donnée est importante pour déterminer si la méthode de génération est assez rapide pour être intégrée dans un environnement de génération de trafic. En particulier, comme cela sera discuté dans la deuxième partie de ce manuscrit (chapitre 6), dans le cadre de la génération de trafic sur puce, l'objectif est de faire des simulations rapides. Il convient donc de s'assurer que le temps de la génération n'est pas trop important.

La figure 3.1 montre l'évolution du temps de génération de réalisation de processus à longue

[1] Gnu's Not Unix

mémoire non-gaussiens (FGN et FARIMA) en fonction de leur taille N (les temps ont été obtenus avec nos routines MATLAB). Le comportement linéaire dans un diagramme log-log montre bien que la génération est polynomiale, la complexité théorique étant $O(N \log N)$.

Pour générer des réalisations de processus suivant une loi marginale Gamma et χ^2, on utilise un nombre de processus gaussiens indépendants, en fonction d'un des paramètres de la loi (M pour la loi du χ^2 et 2α pour la loi Gamma, voir tableau 3.1). On tire néanmoins parti du fait que lors de la synthèse d'un processus gaussien de taille N par la méthode de la matrice circulante, on obtient *deux* processus gaussiens indépendants de même caractéristique (partie réelle et partie imaginaire, voir section 2.8.2). Le temps de génération est donc proportionnel à $N \times M/2$ et $N\alpha$ respectivement pour une loi du χ^2 et une loi Gamma. Ceci explique pourquoi sur la figure 3.1 les courbes Gamma et χ^2 sont légèrement au-dessus des autres, en effet il faut dans les deux cas exécuter deux fois la synthèse par matrice circulante (voir section 2.8.2). Il est aussi à noter que lorsque α augmente, la loi Gamma tend vers une loi Normale (gaussienne). A partir d'un certain seuil, que nous avons expérimentalement fixé à $\alpha_{max} = 50$, il est donc beaucoup plus rapide de générer un processus gaussien, qu'un processus suivant une loi Gamma.

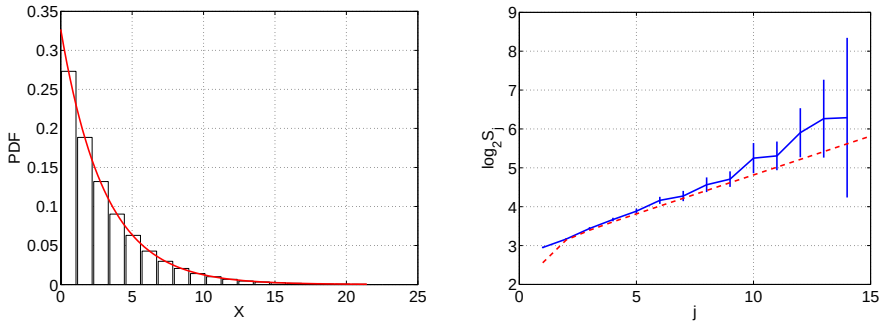
3.2 Validation et performance des estimateurs

Afin de valider la procédure de synthèse, nous avons synthétisé des processus à longue mémoire de paramètre H (FGN et FARIMA) non gaussiens. Nous avons effectué une estimation en ondelettes du paramètre de Hurst H (voir section 2.7.3), et utilisé un maximum de vraisemblance (voir section 1.4.2) pour estimer les paramètres de la PDF. La procédure d'estimation des paramètres du FARIMA sera explicitée dans la section 4.1.4. Nous avons aussi utilisé le test statistique du χ^2 (voir section 1.4.2) pour vérifier que les processus suivaient bien la loi voulue.

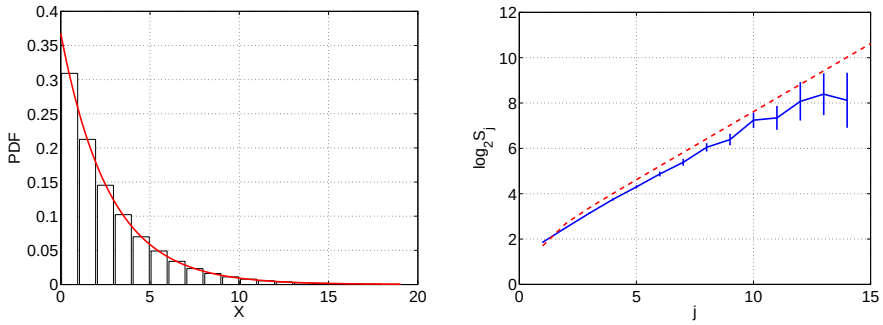
L'évaluation de la performance des estimateurs pour un processus donné (paramètres fixés) correspond à la génération de M réalisations de N échantillons. Pour chaque réalisation, les paramètres du modèles sont estimés. Pour chaque paramètre p du modèle, on note $\hat{p}_i$ la valeur du paramètre p estimée à la réalisation i et simplement p la valeur attendue. On utilise alors différentes statistiques :

- **Biais** de l'estimation : $B(N) = \mathbb{E}(\hat{p}) - p = \frac{1}{M}\big(\sum_{i=0}^{N} \hat{p}_i - p\big)$
- **Biais relatif** en pour cent : $BR(N) = \frac{B(N)}{p} \times 100$
- **Écart type** E : $E^2(N) = \mathbb{E}\big((\mathbb{E}(\hat{p}) - \hat{p})^2\big) = \mathbb{E}(\hat{p}^2) - \mathbb{E}(\hat{p})^2$
- **Erreur quadratique moyenne** : $EQR(N) = \sqrt{E^2(N) + B^2(N)}$

L'évolution de ces statistiques en fonction de N nous permet de mesurer la performance des estimateurs mis en jeu, en particulier leur éventuel biais et leur rapidité de convergence, et si ils ont tendance à sur- ou à sous-estimer les paramètres. Dans tous les résultats qui sont présentés ci-après, le nombre de réalisations a été fixé à $M = 500$. Le biais relatif est représenté en pourcentage dans des diagrammes semi-logarithmiques (seul l'axe des abscisses possède une échelle logarithmique). L'écart type et l'erreur quadratique moyenne sont quant à eux tracés dans des diagrammes log-log (l'axe des abscisses et des ordonnées est en échelle logarithmique), pour pouvoir mieux illustrer la rapidité de converge. En effet, l'écart type décroît (en général) en loi de puissance avec N, ce qui donne une droite dans un diagramme log-log, on peut donc évaluer la rapidité de converge en regardant la pente de la droite.



FGN, H=0.6



FGN, H=0.8

Fig. 3.2: Statistique d'ordre 1 (histogramme empirique, figure de gauche) et statistique d'ordre 2 (diagramme LD, figure de droite) de la génération d'un processus suivant une loi exponentielle et ayant la covariance d'un FGN. Les droites théoriques sont représentées en pointillé.

3.2.1 Résultats pour la génération d'un FGN

Nous présentons ici les résultats de performance d'estimation pour la synthèse d'un processus suivant une loi exponentielle de paramètre $\mu = 3$, et la covariance d'un FGN. Des résultats similaires ont été obtenus avec les autres lois marginales. La figure 3.2 montre la loi marginale et le diagramme LD (voir section 2.7.3). On obtient bien une reproduction conjointe des statistiques d'ordre un (loi marginale) et deux (covariance). En effet l'histogramme empirique se superpose bien sur la PDF attendue, tout comme le diagramme LD empirique se superpose bien sur la droite théorique (celle-ci est toujours incluse dans les barres d'erreurs). Le courbe théorique est calculée à partir de l'expression analytique de la covariance d'un FGN (équation (2.4, section 2.3)).

La figure 3.3 montre les performances d'estimation du paramètre μ de la loi exponentielle (première ligne) et celles du paramètre de longue mémoire H (deuxième ligne). Comme cela est attendu, le biais et l'écart type tendent vers 0 à mesure que N augmente, l'estimation fonctionne donc bien, et il n'y a pas de biais d'estimation (celui-ci convergerait vers une autre valeur que 0). Nous avons utilisé une estimation de μ basée sur un maximum de vraisemblance (MLE) qui fait l'hypothèse que le processus est IID. Il est donc normal que lorsque H augmente – on s'éloigne du comportement IID – la performance de l'estimation soit moins bonne. Néanmoins même pour H proche de 1, le biais et l'écart type de l'estimation restent relativement faibles pour des séries assez longues. Ceci provient du fait qu'il est connu (il peut être démontré) que la loi des grands nombres, qui s'étend aux processus stationnaires au sens large par le théorème de l'ergodicité, voit sa rapidité de convergence décroître de manière significative. En fait, plus les données

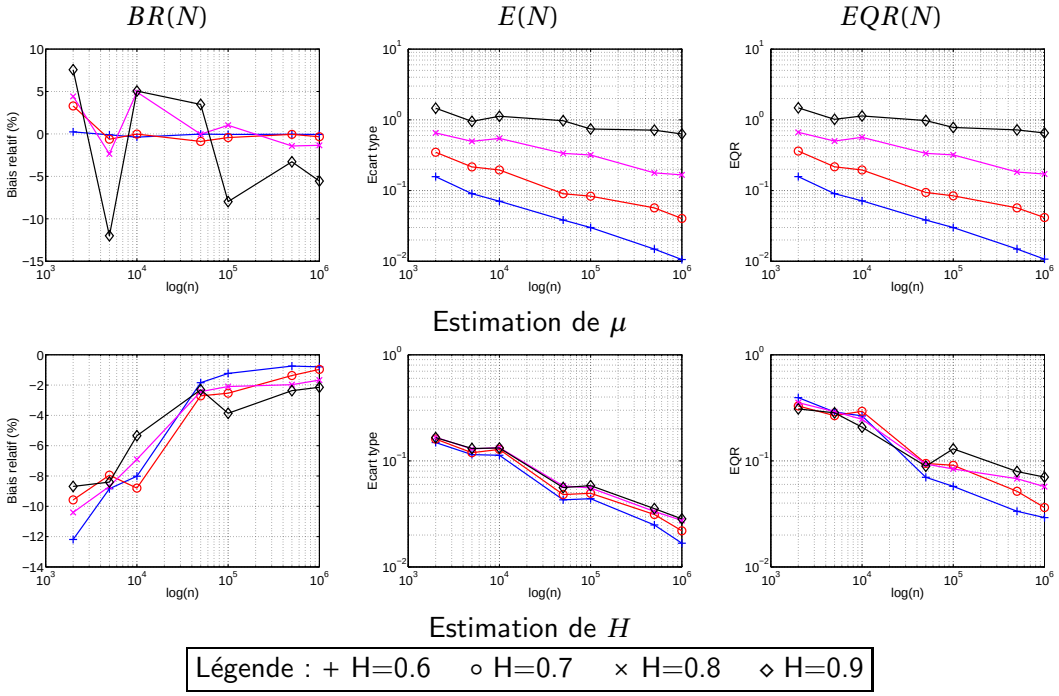


Fig. 3.3: Biais relatif BR, écart type E et erreur quadratique moyenne EQR en fonction de la taille N des réalisations pour l'estimation du paramètre de la loi exponentielle (μ) et du paramètre de longue mémoire (H). Les courbes correspondent à différentes valeurs de H.

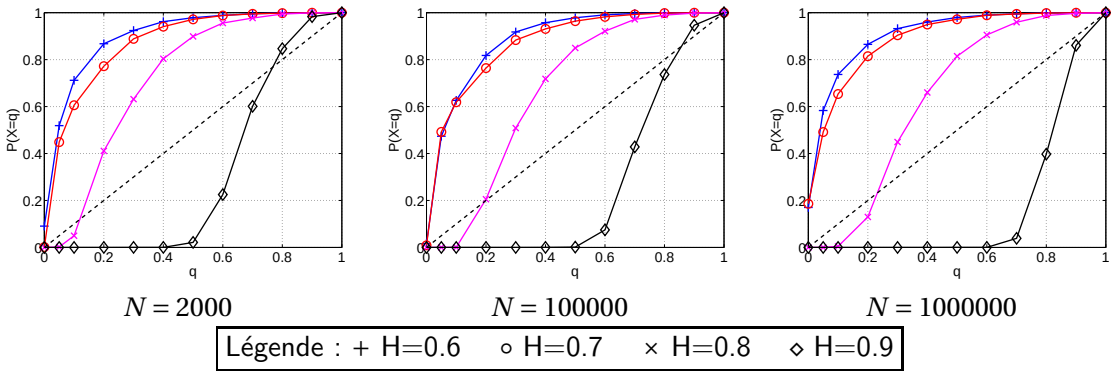


Fig. 3.4: Distribution cumulative de l'indicateur P du test du χ^2 sur les 500 réalisations générées d'un processus suivant une loi marginale exponentielle et ayant la covariance d'un FGN. Les différents graphiques correspondent à différentes tailles de réalisations.

possèdent de la longue mémoire, et plus la convergence des estimateurs standards de moyenne et de variance est lente, les résultats expérimentaux obtenus sont donc en accord avec la théorie.

Les performances d'estimation de H (basée sur la méthode des ondelettes, voir section 2.7.3) sont satisfaisantes. Il est intéressant de remarquer que le biais relatif, si il tend vers 0 quand N augmente, reste toujours négatif, ce qui sous-entend une légère tendance à la sous-estimation de H par l'estimateur en ondelettes pour les petites séries de données. L'écart type de l'estimation de H montre quant à elle une convergence standard.

Pour l'estimation des différents paramètres, l'erreur quadratique moyenne est dominée par l'écart type, ce qui montre que les estimateurs utilisés ne sont pas biaisés.

Pour chaque série synthétique, nous avons effectué le test du χ^2 (voir section 1.4.2, page 27), et en particulier nous avons stocké la valeur de l'indicateur P, compris entre 0 (les données suivent certainement la loi) et 1 (les données ne suivent pas la loi). La figure 3.4 montre la répartition de P pour les $M = 500$ réalisations effectuées pour différentes valeurs de H (sur chaque graphique) et pour trois tailles de réalisations (différents graphiques). Si les données sont IID et suivent bien la loi, alors théoriquement la répartition devrait être uniforme (tracée en pointillé sur les graphiques de la figure 3.4). On peut avoir confiance dans le fait que les réalisations synthétiques suivent bien la loi marginale voulue, car dans le cas contraire l'hypothèse sera systématiquement (ou très souvent) rejetée, ce qui impliquerait que les courbes se réduisent au point supérieur droit des graphiques, c'est à dire dans le cas ou P vaut systématiquement 1. L'écart entre la courbe en pointillé et les courbes obtenues est dû à la présence de longue mémoire dans les données. On peut noter que cet écart varie avec la valeur de H, ce qui peut s'expliquer par le fait que la longue mémoire, qui ralentit la rapidité de convergence de la loi des grands nombres, ralentit de ce fait aussi la convergence de la fréquence d'apparition des valeurs dans une série de données avec sa probabilité. Le test du χ^2 tient compte du nombre de points de la série, c'est pourquoi il est logique qu'il n'y ait pas de différence frappante entre les trois graphiques de la figure 3.4.

3.2.2 Résultats pour la génération d'un FARIMA

Nous présentons ici les résultats de performance d'estimation obtenus pour la génération de processus suivant une loi marginale Gamma de paramètre $\alpha = 2$ et $\beta = 3$, et ayant la covariance d'un FARIMA(1,d,1), avec $\Phi_p(x) = 1 - 0.4x$ et $\Theta_q(x) = 1 - 0.6x$. Le paramètre de longue mémoire d varie entre 0 ($H = 0.5$) et 0.5 ($H = 1$). Des résultats similaires ont été obtenus avec d'autres lois marginales et d'autres polynômes Φ_p et Θ_q.

La figure 3.5 (page 57) montre la loi marginale et le diagramme LD (voir section 2.7.3). On obtient bien une reproduction conjointe des statistiques d'ordres un (loi marginale) et deux (covariance). En effet l'histogramme empirique se superpose bien sur la PDF attendue, tout comme le diagramme LD empirique se superpose bien sur la courbe théorique (celle-ci est toujours incluse dans les barres d'erreurs), comme dans la section précédente. La courbe théorique est calculée à partir de l'expression analytique de la densité spectrale de puissance d'un FARIMA (équation (2.7, section 2.6)).

La figure 3.6 (page 57) montre les performances d'estimation des paramètres α et β de la loi $\Gamma_{\alpha,\beta}$. L'estimation a été faite en utilisant un maximum de vraisemblance (MLE, voir section 1.4.2). Les résultats sont comparables à ceux obtenus pour la génération d'un FGN suivant une loi exponentielle (voir section précédente), le biais converge vers 0 et l'écart type décroît en loi de puissance, avec une convergence d'autant plus lente que le paramètre de longue mémoire H est proche de 1.

La figure 3.7 (page 58) montre quant à elle les résultats du test du χ^2. Les mêmes commentaires que ceux qui ont été faits sur la génération de FGN dans la section précédente sont valables car les courbes sont similaires.

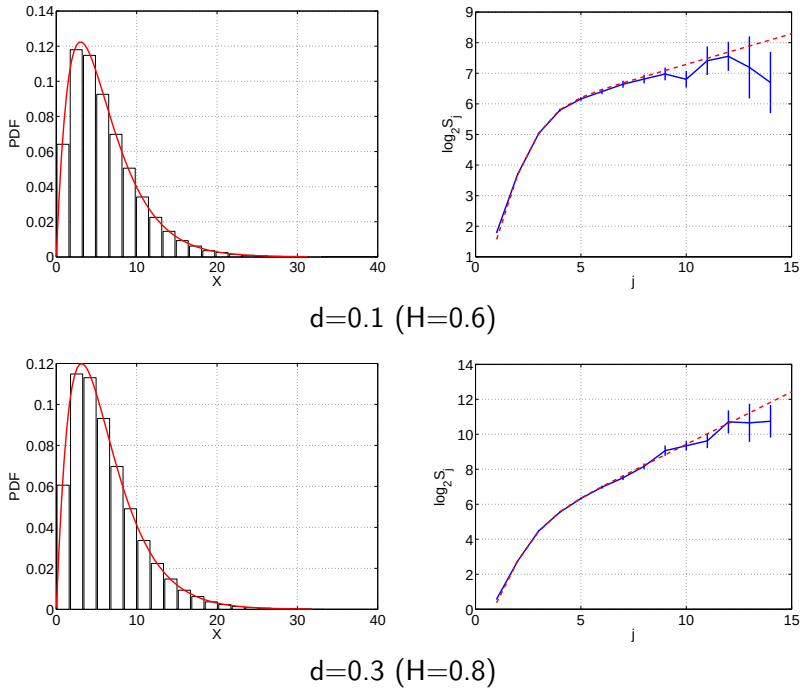


Fig. 3.5: Statistique d'ordre 1 (histogramme empirique, figure de gauche) et statistique d'ordre 2 (diagramme LD, figure de droite) de la génération d'un processus suivant une loi exponentielle et ayant la covariance d'un FARIMA. Les courbes théoriques sont représentées en pointillé.

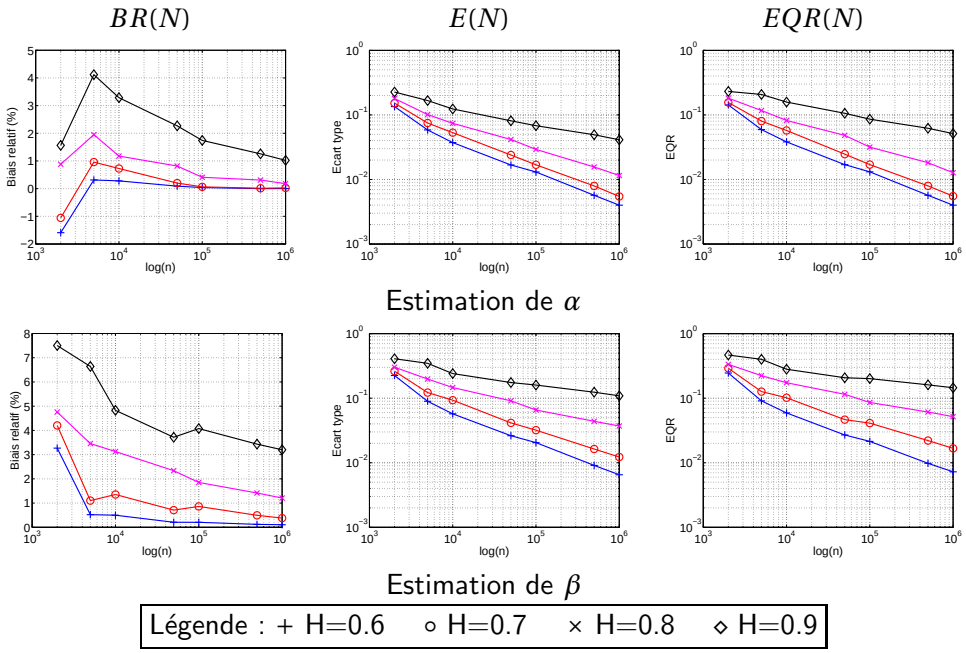


Fig. 3.6: Biais relatif BR, écart type E et erreur quadratique moyenne EQR en fonction de la taille N des réalisations pour l'estimation des paramètres de la marginale pour la génération de processus FARIMA suivant une loi marginale Gamma(α,β).

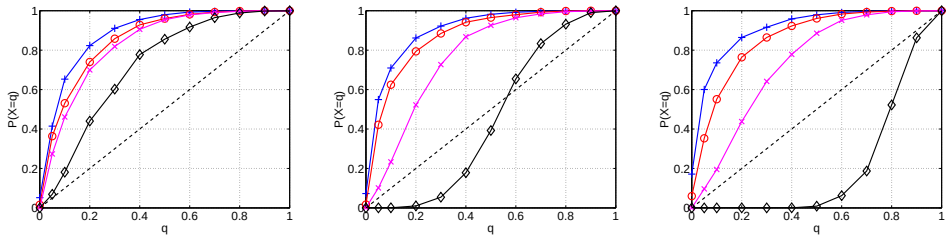


Fig. 3.7: Distribution cumulative de l'indicateur P du test du χ^2 sur les 500 réalisations générées d'un processus suivant une loi marginale Gamma et ayant la covariance d'un FARIMA. Les différents graphiques correspondent à différentes tailles de réalisations.

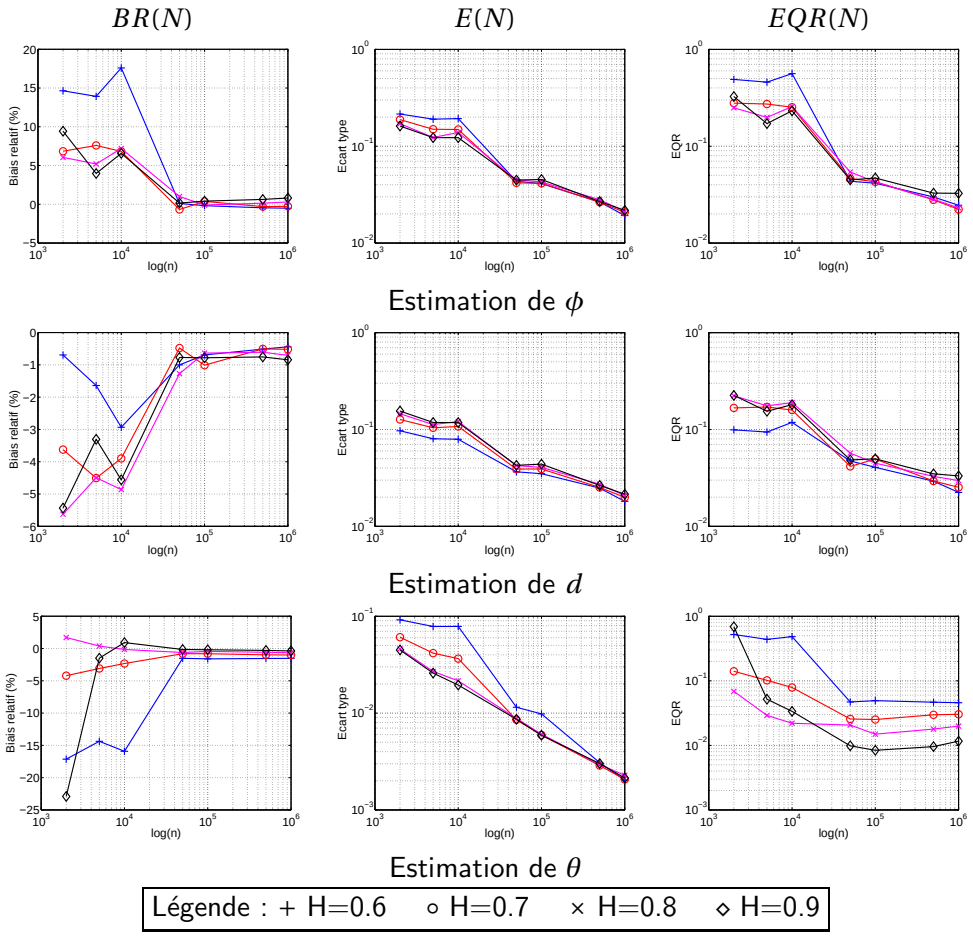


Fig. 3.8: Biais relatif BR, écart type E et erreur quadratique moyenne EQR en fonction de la taille N des réalisations pour l'estimation des paramètres de la covariance pour la génération de processus FARIMA suivant une loi marginale Gamma(α,β).

La figure 3.8 (page 58) montre enfin les performances d'estimation des paramètres ϕ et d et θ de la covariance d'un FARIMA(1,d,1) ($H = d + 1/2$). La procédure d'estimation implique un *blanchiment* des données pour que l'estimation ARMA puisse fonctionner (cette procédure est détaillée dans la section 4.1.4). Cette étape est réalisée avec l'estimation $\hat{H}$ par la méthode des ondelettes (voir section 2.7.3), l'erreur d'estimation de H est donc reportée sur ϕ et θ. Ceci explique pourquoi la performance de l'estimateur ARMA est moins bonne que l'estimation des paramètres de la loi marginale. Néanmoins les résultats restent satisfaisants, et montrent que la covariance est bien exactement reproduite par notre méthode de synthèse, contrairement à la méthode de l'inverse (voir section 3.1.1) qui ne garantit qu'un maintien asymptotique de la covariance, et donc une perte du comportement dans les petites échelles.

On peut enfin remarquer que, comme dans le cas du FGN suivant une loi exponentielle, le biais de l'estimation du paramètre de longue mémoire est toujours négatif, ce qui montre que l'estimateur en ondelettes a tendance à sous-estimer, pour des petites séries, la valeur de H.

3.3 Conclusion

Nous avons, dans ce chapitre, présenté notre contribution à la génération de réalisations de processus à longue mémoire non-gaussiens. Nous avons pu, grâce à cette méthode, étudier la performance des estimateurs de paramètres dans le cas où le processus n'est pas gaussien, ce qui constitue aussi une contribution. Nous disposons maintenant de moyens (algorithmes et programmes) pour générer des réalisations de processus à longue mémoire (covariance d'un FGN et d'un FARIMA), et suivant une large gamme de lois marginales (Gamma, Exponentielle, LogNormal, χ^2, Pareto et Uniforme). Cette méthode pourra être utilisée pour générer du trafic synthétique ayant les caractéristiques statistiques d'ordre un et deux d'un trafic de référence.

Le problème théorique reste néanmoins ouvert, dans le sens où il n'existe pas encore de méthode pour générer de manière exacte une réalisation d'un processus stochastique ayant une loi marginale quelconque et une covariance quelconque. Si notre méthode de synthèse comporte quelques limitations en terme de loi marginale et de forme de covariance, il s'agit néanmoins d'un outil intéressant pour la communauté scientifique.

Fort de cette capacité à prendre en compte de manière conjointe la loi marginale et la covariance d'un processus pour la génération de réalisations de celui-ci, nous avons proposé un modèle adapté aux série de débits agrégé de trafic Internet, qui constitue l'objet principal du chapitre suivant.

Chapitre 4
Contributions de la thèse à l'analyse de trafic Internet

Ce chapitre présente différentes contributions concernant la modélisation de trace de débit agrégé de trafic Internet (section 4.1), un modèle multirésolution dérivé de cette modélisation adapté à la détection d'anomalies et d'attaques (section 4.2) et enfin la simulation de trafic à longue mémoire dans le simulateur de réseau NS-2 (section 4.3).

4.1 Modélisation et synthèse de trafic Internet

La modélisation des séries de télé-trafic informatique, du trafic Internet par exemple, est un exercice obligatoire pour la gestion des réseaux. Celle-ci est, en effet, indispensable pour réaliser des prévisions de performance pertinentes, améliorer le fonctionnement du réseau, y assurer une certaine qualité de service (QoS), en optimiser la gestion ou décider de règles de tarifications. De nombreux types de modélisations ont été proposés, chacune d'entres elles recherchant à caractériser au mieux les propriétés statistiques du trafic Internet au niveau connexion [137, 11, 12, 60], ou au niveau paquet [6, 136].

Le trafic Internet présente deux caractéristiques statistiques principales, unanimement reconnues, qu'il est essentiel de prendre en compte pour réaliser des modélisations efficaces : il est *non gaussien* [104] et à *longue mémoire* [88, 19, 121, 117, 118]. Les travaux s'intéressent souvent à l'une ou l'autre de ces caractéristiques mais peu essaient d'atteindre une description pertinente des deux. Souvent, les travaux qui visent ce double objectif reposent sur une superposition de processus de Poisson modulés par une chaîne de Markov (MMPP, voir section 1.3.3.3, [136, 109, 6]), impliquant un grand nombre de paramètres à estimer afin d'ajuster la distribution marginale et la covariance.

4.1.1 Introduction

Dans ce travail, nous nous intéressons à une modélisation conjointe de la loi marginale et de la structure de covariance de séries de télé-trafic informatique, qui capture en peu de paramètres à la fois leur caractère non-gaussien et leur longue mémoire. Nous proposons l'utilisation d'un processus stochastique non-gaussien à longue mémoire, dont la distribution marginale est une loi Gamma, et dont la structure de corrélation est celle d'un processus FARIMA. À partir de plusieurs jeux de traces de références de télé-trafic informatique, nous montrons comment ce modèle, avec peu de paramètres, capture efficacement les caractéristiques statistiques de premier et second ordre des traces de débit agrégé. Nous observons notamment que cette modélisation reste pertinente pour une très large gamme de niveaux d'agrégation Δ. Aussi nous disposons, grâce à la méthode de synthèse développée dans le chapitre précédent, d'un moyen pour générer des réalisations statistiquement proches des traces de trafic que nous analysons. Disposer de

Trace	Date	Durée (s)	Liaison	NP (10^6)	IAT (ms)
PAUG	29/08/1989	2620	LAN(10BaseT)	1	2.6
LBL-TCP-3	20/01/1994	7200	WAN(10BaseT)	1.7	4
AuckIV	**02/04/2001**	**10800**	**WAN(OC3)**	**9**	**1.2**
CAIDA	14/08/2002	600	Backbone(OC48)	65	0.01
UNC	06/04/2003	3600	WAN	4.6	0.8

Tab. 4.1: Description des traces analysées. La colonne NP correspond au nombre de paquets de la trace et "IAT" est le temps moyen entre deux paquets. La ligne en gras correspond à la trace sur laquelle les résultats expérimentaux sont présentés.

telles procédures est un enjeu essentiel. D'une part, elles permettent d'étudier par simulations numériques les performances de files d'attente et de réseaux, qui ne pourraient être atteintes par calcul analytique du fait des propriétés statistiques non-standard du trafic correspondant. D'autre part, elles peuvent être utilisées pour générer du trafic pour nourrir par exemple des maquettes de réseau afin d'étudier la qualité de leur fonctionnement.

4.1.2 Traces utilisées

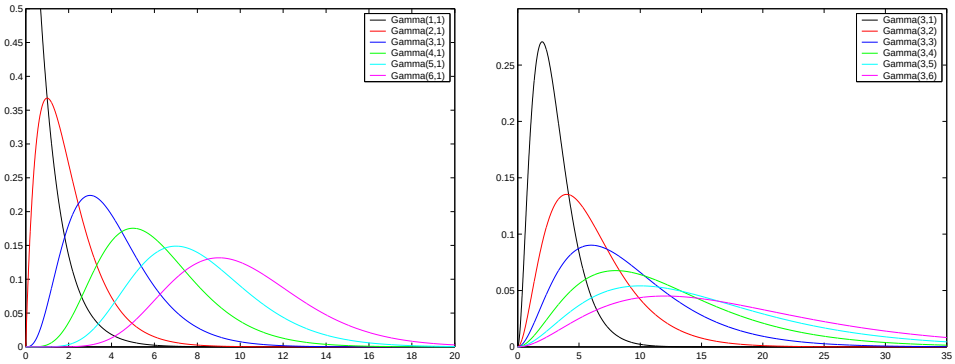
Nous avons travaillé à partir d'une variété de traces de trafic informatique collectées entre 1989 et 2003, correspondant à des types de trafic et de réseaux différents : *Local*, *Metropolitan*, *Wide Area Network*, périphérie ou cœur de réseau, etc. Elles sont rassemblées dans le tableau 4.1. Ces traces sont disponibles sur les sites des principaux groupes de recherche universitaires impliqués (WAND, CAIDA, LBL[47], UNC), comme cela est indiqué dans le tableau 4.2.

Trace	Groupe de recherche	Lien
PAUG	LBL - États-unis	ita.ee.lbl.gov/index.html
LBL-TCP-3	LBL - États-unis	ita.ee.lbl.gov/index.html
AuckIV	WAND - Nouvelle Zélande	wand.cs.waikato.ac.nz/wand/wits
CAIDA	CAIDA - États-unis	www.caida.org/analysis/workload/oc48/
UNC	UNC - États-unis	www-dirt.cs.unc.edu/ts/

Tab. 4.2: Description des traces analysées. Localisation des groupes de recherche impliqués et liens Internet.

Ces traces contiennent de nombreuses informations. Pour chaque paquet, on a l'instant d'arrivé (*timestamp*), l'adresse *IP*[1] source et destination, le port source et destination ainsi que le protocole de niveau transport utilisé (TCP ou UDP). Le trafic Internet circulant sur un lien peut donc être vu comme une séquence de paquets. Le nombre de paquets transitant sur un lien étant très important, les traces occupent rapidement un espace gigantesque, et en pratique il est difficile de faire des analyses statistiques directement sur ces traces. C'est pourquoi nous nous sommes focalisés sur la modélisation des traces de débit agrégé, qui peuvent être facilement extraites des traces complètes. On définit le débit agrégé X_Δ (respectivement W_Δ), comme le nombre de paquets (respectivement octets) transmis dans des fenêtres temporelles successives de taille Δ. $X_\Delta(k)$ est donc le nombre de paquets dont l'instant d'arrivée est compris entre $k\Delta$ et $(k+1)\Delta$. W_Δ tient en plus compte de la taille des paquets. En pratique, dans les traces que nous avons analysées, la taille des paquets étant peu variable, les caractéristiques statistiques de X_Δ et W_Δ

[1] *Internet Protocol*



(a) (b)

Fig. 4.1: Fonction de masse (PDF) des lois Gamma(α,β) pour différentes valeurs de α (a) et β (b).

étaient toujours très proches. Nous ne présenterons donc ici que des résultats sur le nombre de paquets agrégés X_Δ. En général, on extrait de la trace complète une trace de débit agrégé à un niveau d'agrégation initial noté Δ_0, il est alors facile d'obtenir une traces de débit agrégé à $n\Delta_0$:

$$\forall n > 1, \quad X_{n\Delta_0}(k) = \sum_{i=nk}^{n(k+1)} X_{\Delta_0}(i) \tag{4.1}$$

On peut aussi normaliser ce débit en prenant :

$$\forall n > 1, \quad X_{n\Delta_0}(k) = \frac{1}{n} \sum_{i=nk}^{n(k+1)} X_{\Delta_0}(i)$$

Une question centrale dans la modélisation du trafic Internet réside dans le choix d'un niveau d'agrégation Δ pertinent. La solution de cette délicate question dépend à la fois de l'utilisation qui sera faite de la modélisation, de la nature des données et de contraintes techniques. Il est donc essentiel de proposer des modèles qui intègrent naturellement et facilement la possibilité de travailler à différents niveaux d'agrégation. Il faut pour cela que le modèle soit bien représentatif des données et ce pour une large gamme de niveaux d'agrégation.

4.1.3 Modèle proposé

Pour modéliser les séries $\{X_\Delta[n]\}_{n\in\mathbb{N}}$, nous proposons l'utilisation d'un processus stochastique stationnaire suivant une loi marginale Gamma de paramètres α et β et ayant la covariance d'un FARIMA(ϕ, d, θ), et ce pour chaque niveau d'agrégation Δ indépendamment.

• **Marginale non-gaussienne.** La PDF d'une loi marginale Gamma, notée $\Gamma_{\alpha,\beta}$ est :

$$\Gamma_{\alpha,\beta}(x) = \frac{1}{\beta\Gamma_f(\alpha)}(\frac{x}{\beta})^{\alpha-1}\exp(-\frac{x}{\beta}), \tag{4.2}$$

Cette loi est caractérisée par deux paramètres strictement positifs : la forme (α), et l'échelle (β). L'évolution de la loi Gamma (PDF) en fonction de ces deux paramètres est représentée sur la figure 4.1. Elle fournit des variables aléatoires positives, de moyenne $\mu = \alpha\beta$ et de variance $\sigma^2 = \alpha\beta^2$. Le paramètre de forme permet de balayer de manière continue les lois entre la loi exponentielle ($\alpha = 1$) et la loi Normale ($\alpha \to \infty$). L'inverse du paramètre de forme, $1/\alpha$ peut donc

être envisagé comme un indicateur de distance à la loi Normale. La loi Gamma possède de plus la propriété d'être stable sous addition et par multiplication par une constante. Cette propriété est intéressante puisque agréger du trafic correspond justement à sommer et éventuellement à normaliser comme le montre l'équation (4.1).

• **Longue mémoire.** Comme cela est désormais largement admis, le trafic Internet possède une propriété de longue mémoire, mais il possède aussi des dépendances à court terme, dont la structure dépend des mécanismes réseaux mis en œuvre. C'est pourquoi il est naturel d'utiliser un modèle de covariance pouvant rendre compte de ces deux structures : le modèle FARIMA (voir section 2.6). Rappelons qu'un processus FARIMA possède d'une part une structure de covariance contrôlée dans les petites échelles par deux polynômes de degré p et q (Φ_p et Θ_q), et d'autre part une caractéristique de longue mémoire de paramètre de Hurst $H = d + 1/2$. Nous nous limiterons ici à des polynômes de degré 1, c'est pourquoi on notera FARIMA(ϕ, d, θ), ϕ et θ étant alors l'unique coefficient des polynômes Φ_p et Θ_q, et d étant le paramètre de longue mémoire.

Le modèle est donc uniquement composé de cinq paramètres, deux pour la loi marginale Gamma (α et β) et trois pour la covariance (ϕ, d et θ).

4.1.4 Procédure d'analyse

Nous détaillons maintenant les procédures d'analyse et d'estimation des paramètres du modèle proposé, utilisées pour l'étude des séries X_Δ.

• **Paramètres de la loi Gamma.** Les paramètres α et β sont estimés par une procédure standard correspondant à un estimateur de maximum de vraisemblance pour des données IID (voir section 1.4.2). Comme ce maximum de vraisemblance, dans le cas d'une loi Gamma, ne peut pas être calculé analytiquement, une procédure itérative est utilisée pour le calculer. L'initialisation de cette procédure est réalisée à partir des estimateurs standard de moyenne et variance $\hat{\mu}$ et $\hat{\sigma}^2$, selon $\hat{\beta} = \hat{\sigma}^2/\hat{\mu}$ et $\hat{\alpha} = \hat{\mu}/\hat{\beta}$. La performance de cet estimateur en présence de longue mémoire est discutée dans la section 3.2.

• **Paramètre de la covariance.** L'estimation des paramètres de la covariance (ϕ, d, θ) est réalisée en deux étapes. Nous procédons tout d'abord à une analyse en ondelettes de la série X_Δ, afin d'estimer le paramètre de longue mémoire $\hat{d} = \hat{H}/2$ (voir section 2.7.3). Les données sont ensuite *quasiment blanchies* par intégration fractionnaire d'ordre $-\hat{d}$, qui a pour effet de *gommer* la longue mémoire. Concrètement cette étape consiste à prendre la transformée de Fourier de X_Δ, qui doit, si on considère que X_Δ est bien une réalisation d'un processus FARIMA, avoir la forme suivante :

$$TF(\nu) \sim \frac{\sigma^2}{2\pi} \frac{\Phi(e^{-2i\pi\nu})}{\Theta(e^{2i\pi\nu})} (1 - e^{2i\pi\nu})^d \tag{4.3}$$

Pour enlever la longue mémoire, on multiplie cette expression par $(1 - e^{2i\pi\nu})^{-d}$ (il reste alors en théorie les dépendances à court terme, c'est à dire la partie ARMA uniquement). On applique enfin la transformée de Fourier inverse pour obtenir les données blanchies. Cette étape est nécessaire pour que l'estimation des paramètres ARMA, ϕ et θ, ai un sens. On utilise ensuite une procédure d'estimation ARMA (procédure itérative reposant sur l'algorithme de Gauss-Newton) pour estimer θ et ϕ.

Ces procédures sont mises en œuvre sur des portions de données jugées stationnaires. La stationnarité est ici vérifiée expérimentalement en vérifiant la consistance des estimations des paramètres obtenues sur différents morceaux de la trace.

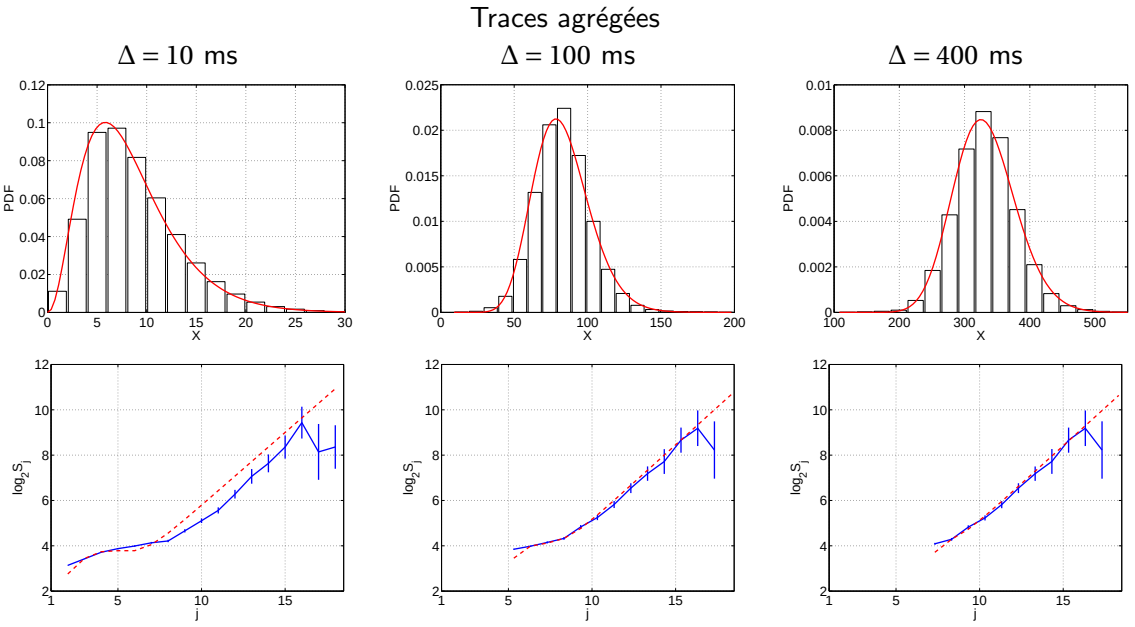


Fig. 4.2: Analyse de la trace AuckIV. Ajustements des histogrammes empiriques par une loi Gamma(α,β) (première ligne) et des covariances par un processus FARIMA$(\hat{\phi},\hat{d},\hat{\theta})$ (deuxième ligne).

4.1.5 Résultats

La figure 4.2 illustre les résultats obtenus sur la trace AuckIV (voir tableau 4.1). Des conclusions identiques ont été trouvées pour toutes les autres traces analysées. La première ligne superposent les marginales empiriques aux lois théoriques $\Gamma_{\hat{\alpha},\hat{\beta}}$ pour les données réelles et synthétiques et la deuxième ligne superpose les diagrammes LD expérimentaux (aussi appelés LD empiriques) à ceux calculés analytiquement en combinant les équations (2.7) et (2.14) pour les données réelles et synthétiques. Ces figures montrent que le modèle $\Gamma_{\alpha,\beta}$ FARIMA(ϕ,d,θ) décrit de façon très satisfaisante les marginales et covariances des données de trafic X_Δ et ce pour une très large gamme de niveaux d'agrégation Δ . En effet l'ajustement aussi bien des histogrammes empiriques que des diagrammes LD est très satisfaisant pour tous les niveaux d'agrégations. Pour la trace considérée (AuckIV), nous avons fixé Δ_0 à 10 ms, et nous avons fait varier Δ entre Δ_0 et 5 secondes. Les figures ne montrent que $\Delta = 10, 100$ et 400 ms. Des résultats similaires ont été obtenus sur toute la gamme considéré. Ce modèle offre donc une modélisation souple et valide pour une très large gamme de niveaux d'agrégation. Ces résultats seront complétés par ceux de la section suivante dans le cadre de la détection d'anomalies.

L'adéquation des lois $\Gamma_{\alpha,\beta}$ aux marginales de X_Δ résulte essentiellement du fait que les lois Gamma constituent une famille stable sous addition (donc par agrégation). Le paramètre α augmente avec Δ, indiquant que les marginales de X_Δ évoluent vers une gaussienne pour les grands Δ ; ainsi la loi $\Gamma_{\alpha,\beta}$ fournit qualitativement une version adaptée aux traces agrégées X_Δ du théorème de la limite centrale (voir section 1.1.2).

La forme des diagrammes LD (deuxième ligne de la figure 4.2) rend compte de l'existence de longue mémoire (un comportement linéaire dans la limite des grandes échelles) ainsi que de celle de corrélation à court terme (décrochement de la droite matérialisant la longue mémoire pour les petites échelles). La pertinence du modèle FARIMA(ϕ,d,θ) se matérialise par le fait que

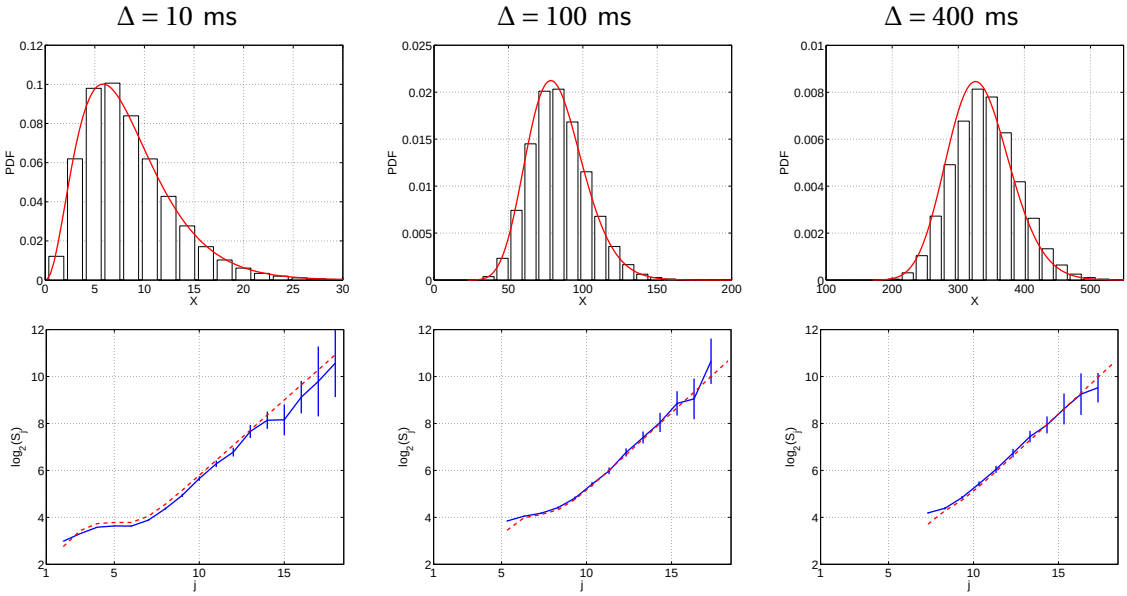


Fig. 4.3: Analyse de traces synthétiques dérivées des ajustements de la trace AuckIV. Ajustements des histogrammes empiriques par une loi Gamma(α,β) (première ligne) et des covariances par un processus FARIMA$(\hat{\phi},\hat{d},\hat{\theta})$ (deuxième ligne).

l'estimation du paramètre de longue mémoire $\hat{d}$ ($H = d + 1/2$) ne varie pas quand Δ évolue, indiquant qu'il rend bien compte d'une propriété de longue mémoire présente dans les données et persistante sous agrégation. Au contraire, les estimés $\hat{\phi}$ et $\hat{\theta}$ décroissent quand Δ augmente, matérialisant le fait que les corrélations à court terme sont, elles, peu à peu gommées par le niveau croissant d'agrégation. La structure de covariance de X_Δ tend alors vers celle d'un processus asymptotiquement autosimilaire, bien approximée par celle d'un FARIMA$(0,d,0)$.

Remarque : la suppression des dépendances à court terme se matérialise en réalité par $\phi = \theta$, en effet dans ce cas les deux parties (AR et MA) s'annulent comme le montre l'équation (2.7 (voir section 2.6)). Ceci sera illustré dans le cadre de la détection d'anomalie dans la section suivante.

La figure 4.3 illustrent les mêmes analyses réalisées sur des traces synthétiques simulées à partir de la procédure décrite plus haut pour des paramètres correspondants à ceux estimés sur la trace réelle de la même ligne. On note que la procédure de synthèse fournit des traces qui possèdent parfaitement les statistiques prescrites comme cela a déjà été mis en évidence dans la section 3.2.

4.2 Détection d'anomalies

La modélisation introduit dans la section précédente a été à la base d'un autre travail effectué dans le cadre du projet METROSEC (métrologie pour la sécurité et la qualité de service dans les réseaux), auquel l'équipe SISYPHE du laboratoire de physique de l'ENS de Lyon participe. L'objectif de ce projet est d'essayer de trouver, dans les caractéristiques statistiques de traces de trafic Internet, une méthodologie pour détecter des attaques et/ou des anomalies dans le réseau. L'idée de base est donc de trouver une modélisation du trafic qui soit valable pour tout type de trafic (avec ou sans anomalies), et d'observer la variations des paramètres du modèle au cours du temps pour détecter des changements de régime.

Ce travail a été principalement réalisée par Pierre Borgnat (Laboratoire de physique, ENS Lyon), Patrice Abry et moi-même pour la partie analyse statistique, et Nicolas Larrieu et Philippe Owesarski (LAAS, laboratoire d'analyse et d'architecture des systèmes) pour la partie mise en place des attaques et collecte des traces. L. Gallon, L. Bernaille, J. Aussibal, G. Dewaele, Y. Zhang, Y. Labit, et de nombreux autres collègues du projet METROSEC ont aussi participé à la mise en place des attaques et à la collecte des traces. Ce travail a été réalisé avec le support du CRI de l'ENS de Lyon. Cette contribution a fait l'objet de publications dans différentes conférences nationales et internationales [SLB+06a, SLB+06b, BLO+06] ainsi que d'un article de synthèse dans le journal *TDSC* [SLB+07].

4.2.1 Introduction

L'Internet est en train de devenir le réseau universel pour tous les types d'informations, du transfert simple de fichiers binaires jusqu'à la transmission de voix, de vidéos ou d'informations interactives en temps réel. L'Internet doit donc évoluer d'une offre de service *best effort* unique vers une offre multi-services, ce qui le rend plus sensible aux attaques, et en particulier aux attaques de déni de service (DOS (pour *Deny of Service*) simples et distribuées (DDOS pour *Distributed* DOS). En effet, les attaques DOS provoquent des changements importants dans les caractéristiques du trafic, ce qui peut réduire de façon significative le niveau de QOS perçu par les utilisateurs du réseau.

Combattre les attaques DOS est une tâche difficile et les systèmes de détection d'intrusion (IDS pour *Intrusion Dectection Systems*), notamment ceux basés sur la détection d'anomalies, ne sont pas efficaces. En premier lieu, leurs limitations sont liées à la multitude de formes que peuvent prendre les attaques DOS qui rendent difficile une définition globale de celles-ci. Dans un second temps, il a été observé à de très nombreuses reprises que le trafic Internet normal présente des variations importantes de son trafic à toutes les échelles [117], souvent décrites en terme de longue mémoire [42], auto-similarité [118], multifractalité [48]. Ces caractéristiques rendent plus délicate et incertaine la détection d'anomalies. Troisièmement, le trafic Internet peut présenter des variations fortes, soudaines mais légitimes (comme des foules subites - *flash crowd* en anglais - par exemple) qu'il peut être difficile de distinguer des variations illégitimes. Pour ces raisons, les IDS utilisant la détection d'anomalies souffrent souvent d'un taux élevé de faux positifs, et sont donc peu populaires. L'évolution actuelle du trafic Internet, avec une variété immense de types de trafics rendent encore plus délicate la conception d'un IDS efficace.

Ce travail a pour objectifs principaux d'analyser l'impact des anomalies sur les caractéristiques statistiques et de mettre en évidence des signatures caractéristiques du trafic contenant des anomalies légitimes et illégitimes. Ces résultats doivent servir à améliorer les mécanismes utilisés dans les réseaux afin de les rendre capable de combattre ces anomalies.

Nous proposons pour cela d'utiliser un modèle de processus stochastique non-gaussien et à mémoire longue représentant le trafic Internet : le modèle GAMMA – FARIMA introduit dans la section précédente, qui montre une capacité à bien représenter le trafic à différents niveaux d'agrégation [SA05]. Nous montrons ici que les évolutions des estimations des paramètres de ce modèle permettent de différencier le trafic avec et sans anomalies et de classifier ces anomalies.

4.2.2 État de l'art

Les IDS basés sur la détection d'anomalies n'utilisent pas, en général, des modèles statistiques riches. Ils supervisent juste des paramètres simples du trafic comme son débit d'octets ou de paquets. La plupart de ces IDS recherchent juste des séquences de paquets spécifiques, connues comme des signatures d'attaques [122]. Les alarmes sont essentiellement générées lorsqu'un seuil est dépassé [26, 149], ce qui conduit à un nombre important de faux positifs [108]. Par conséquent, ces IDS sont souvent assez peu satisfaisants car ils ne peuvent pas différencier les variations légitimes du trafic des attaques.

Les récents résultats obtenus en modélisation par différents projets de métrologie du trafic ont cependant permis d'envisager de nouvelles stratégies de détection d'intrusion. Même si ces dernières sont jeunes et en cours de développement, des résultats très intéressants utilisant des caractérisations statistiques ont été publiés. Par exemple, Ye a proposé un modèle Markovien pour le comportement temporel du trafic [159], et génère des alarmes lorsque le trafic s'éloigne significativement du modèle. D'autres auteurs [71, 163] ont montré que les attaques DOS augmentent la corrélation dans le trafic, ce qui pourrait représenter une technique de détection robuste. A partir de l'évaluation de l'inter-corrélation de trafics sur différents liens, Lakhina *et al.* ont proposé une méthode pour détecter les anomalies dans les matrices de trafic à l'échelle du réseau global [82]. Hussain *et al.* utilisent la densité spectrale de puissance pour identifier des signatures pour différentes attaques [63]. De la même façon, l'estimation spectrale a été utilisée pour comparer des trafics avec et sans attaques [31]. Alors que la densité spectrale met en évidence des pics autour du temps d'aller retour dans le réseau (RTT pour *Round Trip Time*) pour du trafic normal, ces pics disparaissent en cas d'attaque. Cette caractéristique peut ainsi être utilisée pour concevoir de nouveaux IDS. Enfin, Li et Lee ont utilisé des techniques à base d'ondelettes pour calculer une distribution d'énergie. Ils ont observé que cette distribution présente des pics lorsque le trafic contient des attaques qui n'existent pas pour le trafic régulier [90]. Barford *et al.* [14] utilisent quant à eux une analyse multirésolution pour détecter les anomalies du trafic. De nombreux autres travaux prometteurs ont déjà été publiés dans le domaine des attaques DOS [73]. La détection d'anomalies pourrait aussi reposer sur des analyses dépendantes des applications [45], ou sur les mécanismes des attaques [75].

Dans ce travail, nous nous focalisons sur le niveau paquet. La méthode d'analyse proposée est principalement basée sur la détection de changements dans les caractéristiques statistiques du trafic. En comparant les distributions marginales et les fonctions de covariance obtenues d'abord sur du trafic régulier, puis sur des trafics présentant une large variété d'anomalies incluant notamment des anomalies légitimes, nous pouvons différencier les changements du trafic dûs à des actions légitimes d'actions illégitimes.

4.2.3 Attaques et anomalies analysées

De par la difficulté à se procurer des données pour lesquelles des anomalies se produisent, nous avons décidé de réaliser nous-mêmes un ensemble d'expérimentations sur le réseau RE-

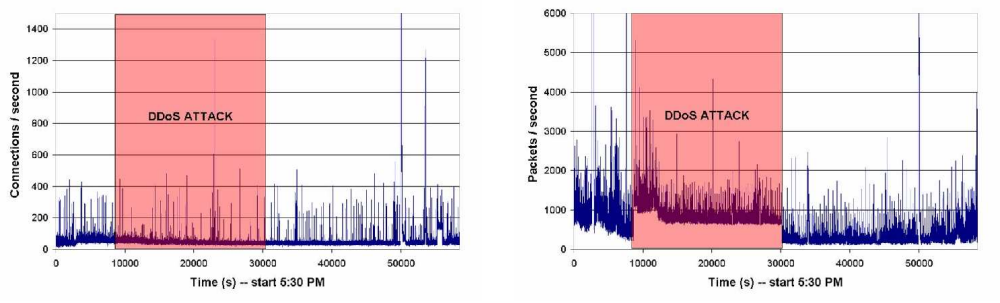
Id	t_i	T(s)	t_a	T_A(s)	D	V	I (%)
flash crowd avec utilisateurs							
FC-1	13 :45	7200	14 :30	1800	-	-	31.27
FC-2	15 :00	7200	15 :45	1800	-	-	18.35
DDoS avec IPERF							
R	17 :30	60000	20 :00	20000	0.5	60	33.82
I	09 :54	5400	10 :22	1800	0.25	1500	17.06
II	14 :00	5400	14 :29	1800	0.5	1500	14.83
III	16 :00	5400	16 :29	1800	0.75	1500	21.51
IV	10 :09	5400	10 :16	2500	1.0	1500	33.29
V	10 :00	5400	10 :28	1800	1.25	1500	39.26
A	14 :00	5400	14 :28	1800	1	1000	34.94
B	16 :00	5400	16 :28	1800	1	500	40.39
C	10 :03	5400	10 :28	1800	1	250	36.93
X	14 :00	5400	14 :28	1800	5	1500	58.02
DDoS avec TRINOO							
tM	18 :21	5400	18 :58	601	0.1	300	4.64
tN	18 :22	3600	18 :51	601	0.1	300	15.18
tT	18 :22	3600	18 :51	601	8	300	82.85

Tab. 4.3: Description des traces contenant des anomalies. R et FC-1 correspondent respectivement à l'attaque DDoS et à la *flash crowd* de référence utilisées pour valider notre modèle. Partie supérieure : attaques DDoS avec IPERF en 2004 et 2005 ; partie inférieure : attaques DDoS avec TRINOO en 2006. Pour chaque trace/attaque, t_i, t_a, T, T_A correspondent respectivement aux temps de début, temps de fin, durée totale et durée de l'attaque (en secondes). D, V et I correspondent respectivement au débit contrôlé de chaque source d'attaque (Mbps), à la taille des paquets d'attaque (en octets) et à l'intensité relative de celle-ci (rapport entre la somme des débits d'attaque et le débit moyen sur le lien vers le LAAS).

NATER (le réseaux des universités françaises), dans le cadre du projet METROSEC. Ces expérimentations incluent la génération d'anomalies légitimes (foules subites) et illégitimes (attaques DDOS). Ceci nous permet de réaliser des expérimentations d'une façon reproductible, précise et contrôlée. L'ensemble des traces est présenté dans le tableau 4.3. Le fait de pouvoir contrôler de manière précise le comportement des attaques (et ainsi de pouvoir reproduire celles-ci) est une étape cruciale pour le développement de système de détection efficace.

• **Attaques DDOS.** Les attaques DDOS étudiées ici sont de type inondation UDP distribuées[2] (*distributed* UDPflooding *en anglais*). Elles ont été générées depuis 5 sites différents : l'IUT De Mont de Marsan, le LIAFA Paris, l'ENS Lyon, l'ESSI Nice, en France et l'université de Coimbra au Portugal, à l'encontre du LAAS à Toulouse qui était le site ciblé. Deux types de génération d'attaques ont été utilisés : IPERF [66] et TRINOO [148]. TRINOO utilise un démon, ce qui permet d'effectuer des attaques plus complexes et donc plus réalistes. Le LAAS est connecté à RENATER par l'intermédiaire d'un lien Ethernet 100 Mbps qui n'a pas été saturé pendant l'attaque. Les caractéristiques de base du trafic de l'attaque notée R dans le tableau 4.3 sont décrites sur les

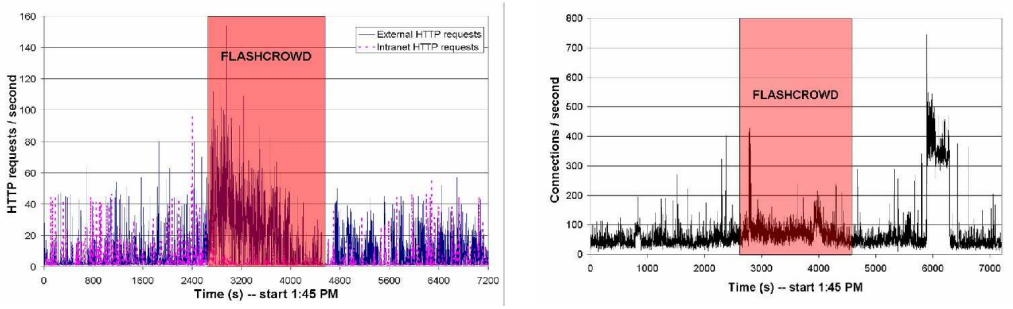
[2] *User Datagram Protocol*



(a) (b)

Fig. 4.4: Débit paquets (a) et Nombre de connexions par seconde (b) pour l'attaque DDoS de référence (R).

Figures 4.4(a) et 4.4(b) qui montrent respectivement le nombre de flux et de paquets sur le réseau d'accès du LAAS. Alors que le premier reste très stable, le second présente une augmentation significative du débit des paquets (il est multiplié par presque 3 pendant l'attaque).



(a) (b)

Fig. 4.5: Nombre de requêtes HTTP (a), et nombre de connexions (b) pour la *flash crowd* de référence (FC-1).

• ***flash crowd*** **(FC) ou foule subite.** Pour comparer l'impact sur les caractéristiques du trafic d'attaques DDoS et de variations légitimes du trafic, nous avons créé des foules subites sur un serveur WEB. Pour les rendre réalistes, c'est à dire humainement aléatoires, nous avons choisi de ne pas utiliser un programme automatique, mais au contraire, de demander à de nombreux collègues de consulter le site WEB du LAAS (http://www.laas.fr). Les résultats présentés sont ceux obtenus pour la foule subite du 14 avril 2005 qui a duré 30 minutes et a rassemblé plus de 100 participants. La figure 4.5(a) montre le nombre de requêtes reçues par le serveur WEB du LAAS, en faisant la distinction entre les requêtes venant de l'intérieur et de l'extérieur du LAAS. Il apparaît clairement que de nombreux utilisateurs ont commencé à naviguer sur le site WEB du LAAS à 14h30 (augmentation importantes du nombre de requêtes), mais également que la plupart ne sont pas restés les 30 minutes. La figure 4.5(b) montre le nombre de flux TCP. Comme c'était attendu, la courbe présente une augmentation du nombre moyen de flux pendant la foule subite. Cependant, il apparaît aussi une augmentation importante du nombre de flux après la fin de l'expérience de foule subite. Pour comprendre ces augmentations, nous avons analysé différentes composantes du trafic en utilisant l'outil *Trafic Designer* de la société QoSMOS [38]. L'analyse a montré que l'augmentation autour de 14h (avant notre expérience) est due à des membres du LAAS qui naviguent sur le WEB juste après le déjeuner. Un tel comportement a été observé systématiquement sur toutes les traces collectées au LAAS depuis. Le second pic, après l'expérience,

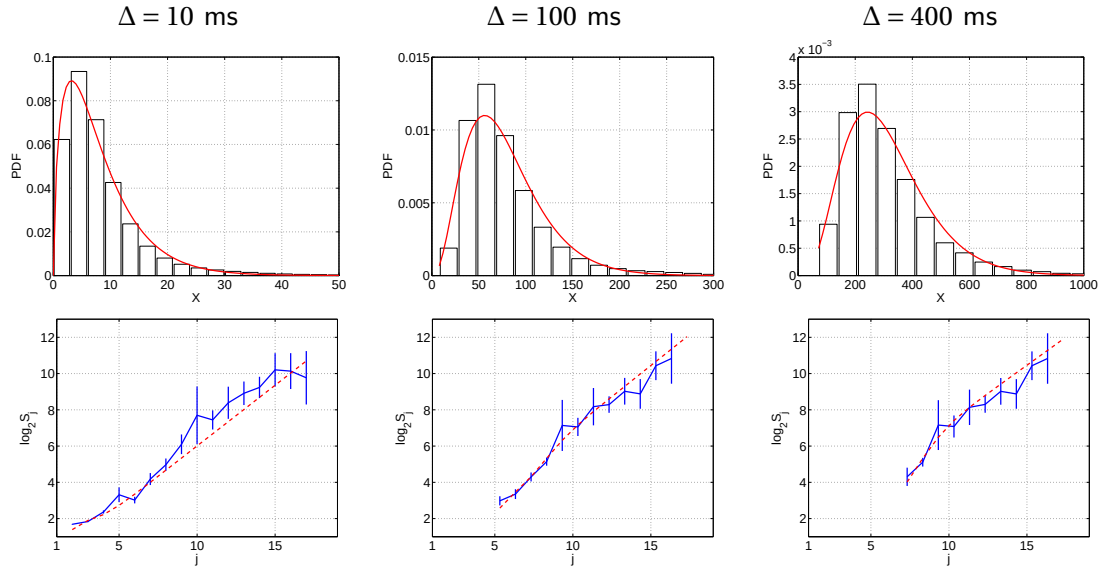


Fig. 4.6: Trace METROSEC-ref1. Ajustement $\Gamma_{\alpha,\beta}$ - FARIMA(ϕ,d,θ) pour les marginales (première ligne) et les covariances (deuxième ligne) pour $\Delta = 10,100,400$ ms. $j = 1$ correspond à 10 ms.

est dû à du trafic SMTP. Il peut s'expliquer de deux façons. En premier lieu, il faut savoir que de nombreux chercheurs au LAAS utilisent l'application *webmail*. Comme le serveur a été très ralenti pendant l'expérience de foule subite, ils ont donc arrêté d'envoyer des e-mails jusqu'à ce que le serveur recommence à fonctionner avec des performances satisfaisantes. Dans un second temps, il faut savoir que le mécanisme de *grey listing* (utilisé pour réduire le nombre de *spam*[3]) retarde certains e-mails, et les émet tous ensemble lors des ouvertures planifiées des portes. La première ouverture après l'expérience s'est produite à 15h15.

4.2.4 Modèle de trafic multirésolution

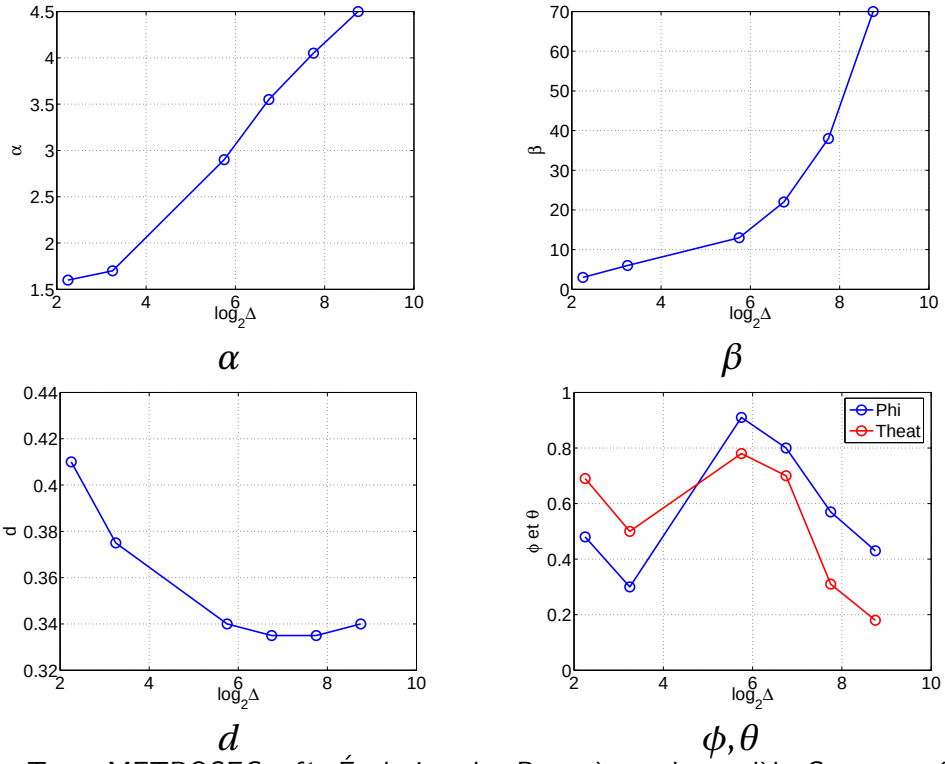
Nous avons utilisé le même modèle que celui présenté à la section 4.1.3 (loi marginale Gamma et covariance d'un FARIMA) pour modéliser les traces de débit agrégé X_Δ. La pertinence de ce modèle pour la modélisation de trace de trafic provenant des grands répertoires de traces mondiaux a été discutée dans la section 4.1.5. Il a été notamment mis en évidence que ce modèle est valable pour une large gamme de niveaux d'agrégation Δ. Nous avons utilisé ce modèle sur des traces de trafic contenant des anomalies.

L'idée est ici non plus de fixer Δ, mais de s'intéresser aux variations des paramètres de modèle en fonction de Δ. Nous allons montrer dans la section suivante que cela nous permet de mettre au point un système de détection d'anomalies robuste.

4.2.4.1 Trafic régulier

• **Lois marginales** La figure 4.6 (première ligne) montre les histogrammes empiriques ainsi que l'ajustement par une loi $\Gamma_{\alpha,\beta}$. Comme cela a déjà été montré à la section 4.1.5, la loi Gamma est valable pour une large gamme de niveaux d'agrégation. La figure 4.7 (première ligne) montre cette fois l'évolution des paramètres de la loi Gamma (α et β) en fonction de $\log_2 \Delta$. Rappelons

[3] Courrier indésirable



α β d ϕ, θ

Fig. 4.7: Trace METROSEC-ref1. Évolution des Paramètres du modèle Gamma – farima en fonction de $\log_2 \Delta$ (Δ en millisecondes).

que les propriétés de stabilité par addition et multiplication par une constante font que le comportement en fonction de Δ dans le cas IID (pas de dépendance), peut être facilement calculé. α devrait augmenter linéairement avec Δ et β devrait rester constant. On peut noter que l'évolution observée est loin du comportement IID (ce qui est normal puisque le processus possède des dépendances à court terme et de la longue mémoire). Une analyse fine montre que $\hat{\alpha}(\Delta)$ n'augmente pas lorsque Δ est petit, puis augmente comme $\log_2 \Delta$ pour Δ plus grand. Le comportement $\hat{\beta}(\Delta)$ est proche d'une augmentation en loi de puissance.

• **Covariances** La figure 4.6 (seconde ligne) montre les diagrammes LD empirique ainsi que l'ajustement par la covariance d'un FARIMA(ϕ, d, θ). De même, le modèle FARIMA permet de capturer à la fois les dépendances courtes et la longue mémoire et offre une bonne modélisation des traces de trafic agrégé, comme cela a déjà été montré à la section 4.1.5. On remarque que la longue mémoire est prédominante à partir de l'échelle $j = 10$, c'est à dire environ $1s$, ce qui nous permet de définir une échelle caractéristique pour séparer le comportement dans les grandes échelles de temps de celui dans les petites échelles de temps. L'évolution des paramètres du FARIMA illustre bien le fait que l'agrégation *gomme* petit à petit les dépendances à court terme. On peut en effet voir sur la figure 4.7 que d reste constant (la longue mémoire reste inchangée), alors que ϕ et θ décroissent lorsque Δ augmente. Il est à noter que si $\phi = \theta$ est équivalent à $\phi = 0$ et $\theta = 0$, et on a alors un FARIMA$(0, d, 0)$. Ceci est cohérent avec la théorie car on peut montrer [19] que lorsque le niveau d'agrégation augmente, le FARIMA(ϕ, d, θ) tend vers un FARIMA$(0, d, 0)$.

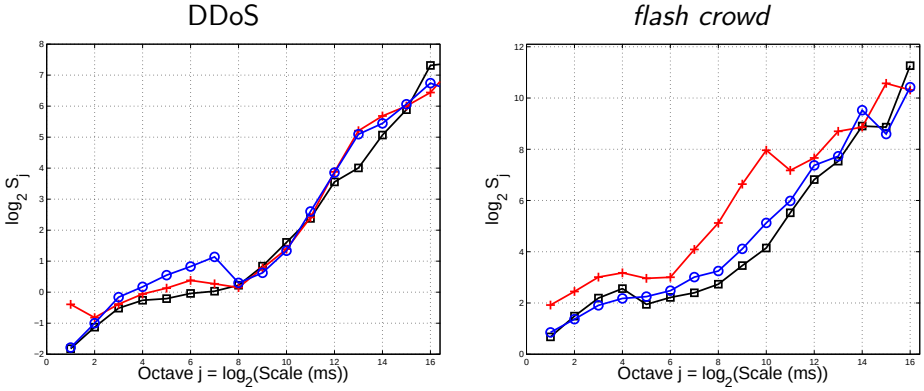


Fig. 4.8: Diagrammes LD pour la DDoS (gauche) et la *flash crowd* (droite). Pour chaque graphique, les courbes montrent pendant l'anomalie (croix), avant (carrés) et après (cercles).

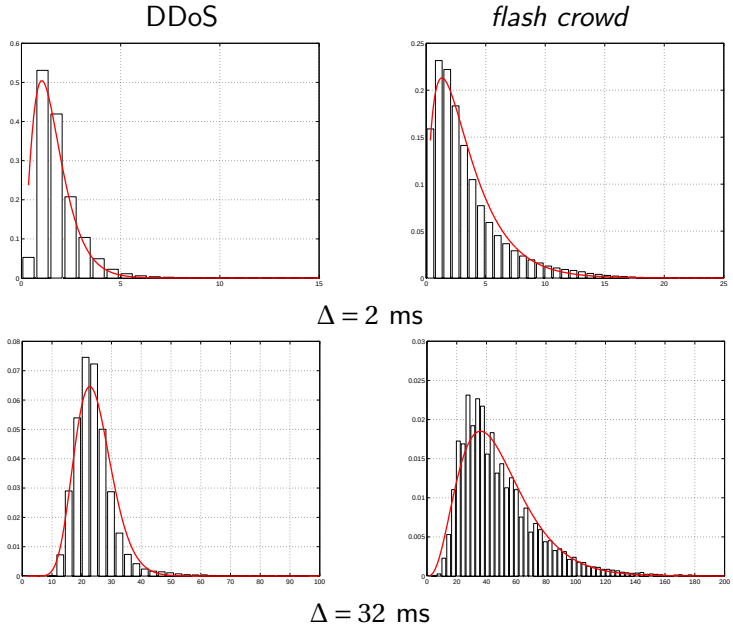


Fig. 4.9: Lois marginales pendant une attaque DDoS (gauche) et une *flash crowd* (droite). Histogrammes empiriques de X_Δ et l'ajustement $\Gamma_{\alpha,\beta}$ pour $\Delta = 2$ ms (haut) et $\Delta = 32$ ms (bas).

4.2.4.2 Trafic avec attaques DDoS

• **Lois marginales.** La figure 4.9 (colonne de gauche) illustre le fait que la loi $\Gamma_{\alpha,\beta}$ est une bonne modélisation pour le trafic avec anomalie car la loi Gamma se superpose bien sur l'histogramme empirique. L'évolution de $\hat{\alpha}$ et $\hat{\beta}$ pendant l'attaque est comparée sur la figure 4.10 (colonne de gauche) à l'évolution avant et après l'attaque. Les estimations sont faites sur des périodes de quinze minutes. Il est clairement mis en évidence sur cette figure que le comportement pendant l'attaque est très différent de ceux avant et après. L'attaque provoque une augmentation rapide de $\hat{\alpha}$, lorsque Δ est très petit, alors que dans des conditions normales, $\hat{\alpha}$ n'augmente pas pour les petits Δ. Cette évolution peut s'expliquer de la manière suivante : pendant l'attaque, un grand nombre de paquets est émis le plus rapidement possible, la probabilité d'avoir 0 paquet dans une fenêtre de taille Δ décroît très rapidement vers 0 lorsque Δ augmente, ce qui n'est pas du tout le comportement du trafic régulier (comparer les figures 4.6 et 4.9). Ceci expliquerait que

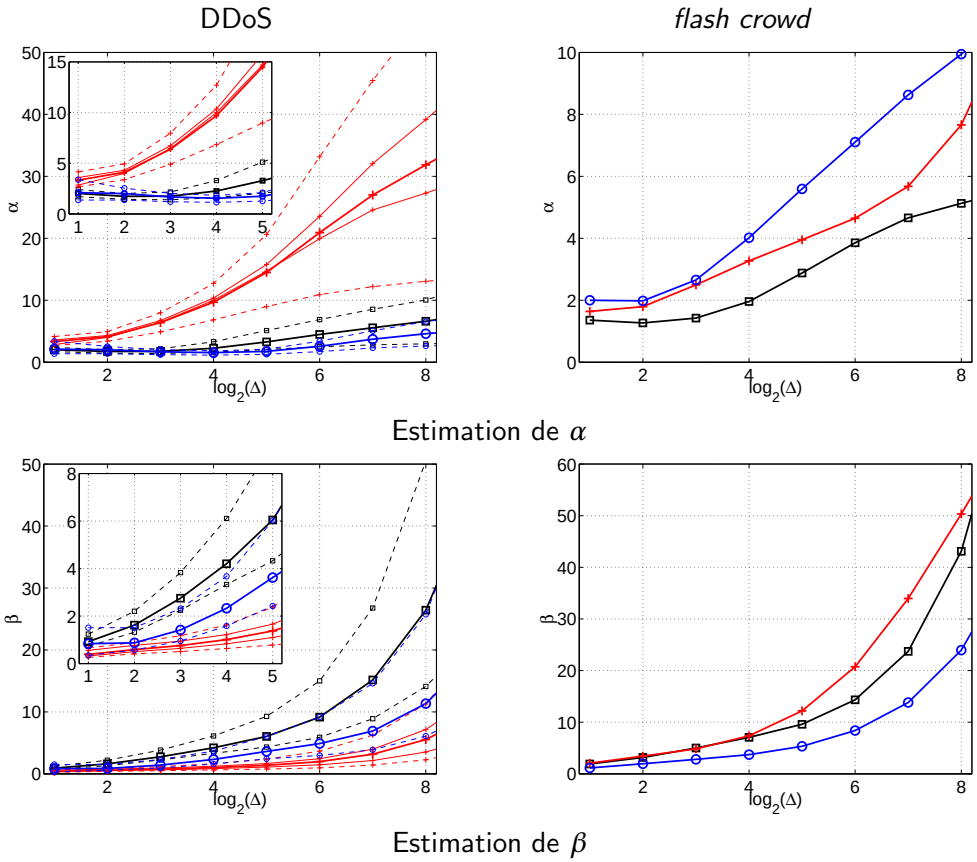


Fig. 4.10: Ajustement $\Gamma_{\alpha,\beta}$. Évolution de $\hat{\alpha}$ (haut) et $\hat{\beta}$ (bas) en fonction de $\log_2 \Delta$ pour une attaque DDoS (gauche) et une *flash crowd* (droite). Pour chaque graphique, les courbes montrent pendant l'anomalie (croix), avant (carrés) et après (cercles). Pour l'attaque DDoS, la moyenne de l'évolution par bloc de 15 minutes est tracée en trait plein avec les valeurs extrêmes de cette évolution en pointillé. Un zoom sur les petits Δ est inséré. Pour la *flash crowd*, qui dure moins longtemps, une estimation sur 15 minutes est présentée pour chaque période (avant, pendant et après).

la croissance de α lorsque Δ est petit est beaucoup plus lente pour le trafic régulier que pour le trafic avec une attaque DDoS. Ceci est équivalent à dire que les lois marginales du trafic régulier tendent beaucoup plus lentement vers une loi Normale (α grand), que les lois marginales du trafic avec une attaque. Cette observation constitue une propriété statistique robuste pour différencier le trafic régulier du trafic sous attaque DDoS.

• **Covariances.** La figure 4.8 (gauche) montre trois diagrammes LD correspondants à trois morceaux de la trace : un pendant l'attaque, un avant et un après. Chaque morceau dure une heure. On peut tout d'abord noter que le modèle FARIMA(ϕ,d,θ) fournit toujours une bonne modélisation du trafic car le diagramme LD théorique se superpose bien sur celui des données. D'autres courbes non présentées ici montrent que cela reste vrai pour une large gamme de niveaux d'agrégation. Aussi, la forme de la covariance pour des échelles de temps supérieures à une seconde ($j = 10$) reste inchangée. En particulier, d, le paramètre de longue mémoire, reste constant. Ceci montre que la longue mémoire est une propriété du trafic régulier, et que cette propriété n'est pas affectée par une attaque DDoS. L'augmentation des corrélations à court terme (pour j compris entre 4 et 7) après l'attaque est dû au fait que cette partie de la trace correspond à une période de nuit, où le trafic circulant sur le réseau est moins important d'une part

et différent d'autre part. Le comportement dans les grandes échelles n'est pas influencé par ce trafic de nuit, renforçant le fait qu'il s'agit bien là d'une caractéristique robuste du trafic qui n'est pas affecté par un changement de volume ou de forme. On peut donc dire qu'il n'est pas possible de différencier le trafic régulier du trafic avec attaque DDoS en se basant sur le digramme LD des traces.

4.2.4.3 Trafic avec *flash crowd*

• **Loi marginales.** La figure 4.9 (colonne de droite) illustre le fait que la loi $\Gamma_{\alpha,\beta}$ est une modélisation pertinente aussi pour ce type de trafic. La figure 4.10 (colonne de droite) montre les courbes $\hat{\alpha}(\Delta)$ et $\hat{\beta}(\Delta)$. On peut remarquer que le comportement pendant la *flash crowd* n'est pas significativement différent de celui avant et après. Ceci est cohérent avec le fait qu'une *flash crowd* n'implique pas de mécanisme qui diminue grandement la probabilité de l'évènement 0 paquet dans une fenêtre temporelle de taille Δ. $\hat{\alpha}(\Delta)$ ne permet donc pas d'identifier une *flash crowd*.

• **Covariance.** La figure 4.8 (droite) montre le diagramme LD pour deux bloc de quinze minutes pendant la *flash crowd*, ainsi que des blocs de quinze minutes enregistrés avant et après l'événement. On note ici que le LD possède un changement important pendant la *flash crowd*. Entre $j = 8$ et $j = 10$, c'est à dire pour des échelles de temps entre 250 ms et 1 s, un important pic d'énergie apparaît. Une telle propriété n'est jamais présente sur l'ensemble des traces que nous avons analysé, et cela pourrait donc être un moyen de détecter un événement de ce type. Le modèle FARIMA ne pourra pas intégrer ce pic, et un test statistique de validité de ce modèle le rejetterait probablement. Ceci peut être la base d'un outil de détection de ce type d'événement. On peut enfin noter que d, le paramètre de longue mémoire, ne change pas significativement pendant la *flash crowd*, la longue mémoire est donc bien une caractéristique propre du trafic Internet, qui n'est pas modifiée par l'apparition d'un événement (*flash crowd* ou attaque DDoS).

4.2.4.4 Résumé et discussions

Voici un résumé des conclusions qui peuvent être tirées des expérimentations présentées dans les trois sections précédentes :

- Le modèle $\Gamma_{\alpha,\beta}$ - FARIMA(ϕ, d, θ) reste un bon modèle pour tous les types de trafic analysés (régulier, avec une attaque DDoS, avec une *flash crowd*).
- Ce modèle est valable pour une large gamme de Δ et nous permet donc de définir un modèle multirésolution en enregistrant l'évolution des paramètres en fonction de Δ. Ceci permet aussi de s'affranchir du choix de Δ, qui un problème difficile dans un cadre d'un système de détection de comportement. Cela nous permet aussi de s'affranchir des valeurs même des paramètres qui changent facilement en fonction de la charge moyenne du lien, etc. En se basant sur la forme de l'évolution des paramètres, on construit donc un système de détection plus robuste. Il est aussi à noter que des graphes d'évolution de la moyenne et de la variance du processus en fonction de Δ ne permettent pas de différencier un trafic régulier d'un trafic avec attaque. C'est bien dans la forme de la distribution (qui est contrôlé, dans le cas d'une loi Gamma, par le paramètre α), qu'il faut chercher la discrimination.
- Les *flash crowd* peuvent être identifiées dans la forme du diagramme LD, cependant ce résultat est trop peu robuste pour être généralisé (deux expérimentations seulement ont été effectuées).
- Ce modèle multirésolution permet d'identifier une attaque de type DDoS dans la forme

de l'évolution de α lorsque Δ est petit. Une *flash crowd* ne sera pas détectée par cette méthode, ce qui est le but recherché. Une procédure de détection est détaillée dans la section suivante.

4.2.5 Procédure de détection

A partir des résultats expérimentaux décrits dans la section précédente, une méthode de détection des attaques DDoS peut être mise en place (nous ne disposons pas assez de traces avec des *flash crowd* pour étudier en détail le problème de la détection de celles-ci). La méthode de détection est comme suit. La trace est découpée en blocs adjacents de taille T. Pour chaque bloc et pour chaque niveau d'agrégation, on calcule une *distance* entre une statistique mesurée sur un bloc de référence, et la même statistique calculée sur le bloc étudié. Cette distance peut être ensuite seuillée pour détecter un comportement anormal et faire remonter une alarme. Comme le suggèrent les expérimentations de la section précédente, nous avons utilisé les statistiques $\hat{\alpha}_\Delta$ et $\hat{\beta}_\Delta$ pour effectuer la détection.

La fenêtre de référence est un bloc de T_{Ref} minutes pris avant les attaques qui peut donc être assimilé à du trafic régulier. Pour le $l^{\text{ième}}$bloc (qui commence à lT), nous avons donc calculé comme expliqué précédemment, $\hat{\alpha}_\Delta(l)$ et $\hat{\beta}_\Delta(l)$.

De nombreuses distances peuvent être utilisées pour obtenir un score qui traduise la proximité ou la divergence de deux distributions. Le lecteur intéressé est renvoyé à l'article [15] qui propose une bonne revues des différentes distances existantes. On pourrait par exemple utiliser une distance non-paramétrique comme la divergence de Kullback [15] pour les lois marginales. Mais on ne tirerait alors pas parti de l'aspect multirésolution de notre modèle. C'est pourquoi nous avons utilisé une distance simple (mais robuste) : la distance quadratique moyenne (MQD pour *Mean Quadratic Distance*) pour $\hat{\alpha}_\Delta(l)$ et $\hat{\beta}_\Delta(l)$. Cette distance est définie ainsi :

$$D_\alpha(l) = \frac{1}{J}\sum_{j=1}^{J}\left(\hat{\alpha}_{2^j}(l) - \hat{\alpha}_{2^j}(ref)\right)^2, \tag{4.4}$$

$$D_\beta(l) = \frac{1}{J}\sum_{j=1}^{J}\left(\hat{\beta}_{2^j}(l) - \hat{\beta}_{2^j}(ref)\right)^2. \tag{4.5}$$

A partir de cette distance, on peut définir un seuil au-delà duquel la divergence par rapport au bloc de référence sera considérée comme signature d'une attaque DDoS. Nous avons fait varier la valeur de ce seuil pour obtenir des courbes expérimentales de performance de cette méthode de détection.

4.2.6 Résultats sur la méthode de détection

La distance par bloc est illustrée sur la figure 4.11 pour une trace contenant une attaque DDoS (Iperf-III sur la figure). On peut voir que la distance sur α ($D_\alpha(l)$) prend de grandes valeurs pendant l'attaque, comme cela était attendu. Ceci montre bien que la présence de l'attaque modifie le comportement de α en fonction de Δ, ce qui augmente la valeur de la distance calculée. Au contraire, la distance sur β ($D_\beta(l)$) reste assez constance, même pendant l'attaque. C'est donc bien dans l'évolution de la *forme* (paramètre α) de la loi marginale avec Δ que se situe un moyen de détecter une attaque DDoS. En effet β, qui est un paramètre d'échelle, va surtout être sensible aux variations de débit sans changement de la structure de corrélation (covariance). Les grandes valeurs présentes dans le graphe de $D_\beta(l)$ sont le fait de blocs à cheval sur le début ou

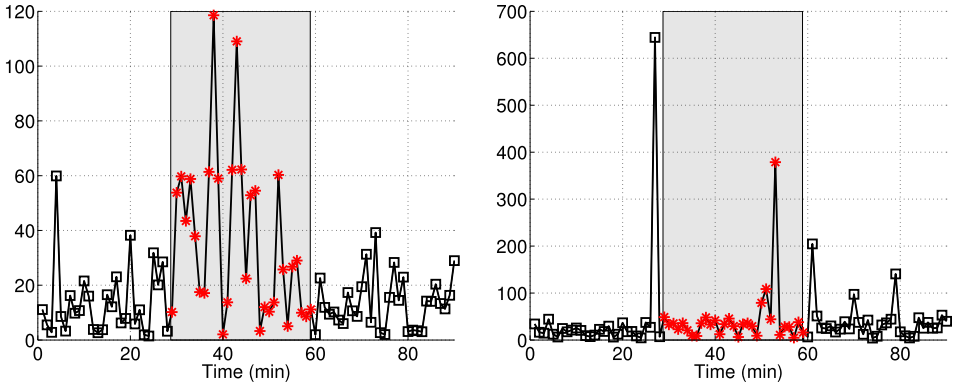


Fig. 4.11: Distance quadratique moyenne (MQD) pour du trafic contenant une attaque DDoS. $D_\alpha(l)$ (gauche) et $D_\beta(l)$ (droite), calculée sur des blocs adjacents de 1 minute. La période d'attaque est indiquée par la partie grisée.

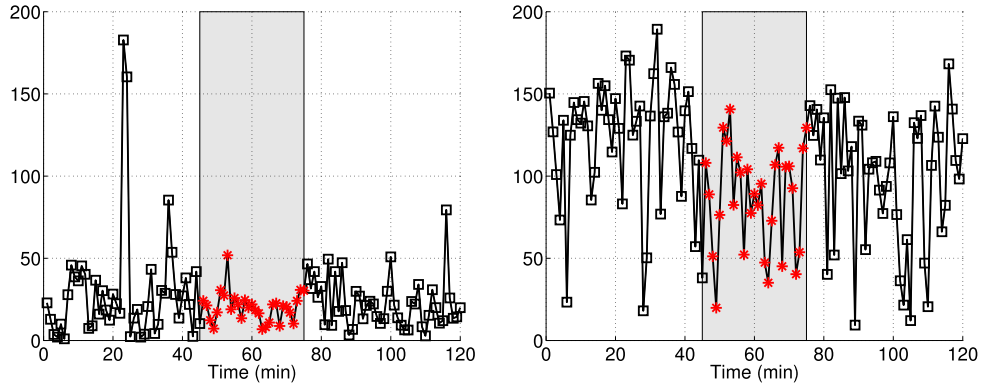


Fig. 4.12: Distance quadratique moyenne (MQD) pour du trafic avec *flash crowd*. $D_\alpha(l)$ (gauche) et $D_\beta(l)$ (droite), calculée sur des blocs adjacents de 1 min. La période de *flash crowd* est indiquée par la partie grisée.

la fin de l'attaque. Ce sont donc des blocs non-stationnaires où l'estimation est aberrante (le test du χ^2 échoue sur ces blocs).

La même distance est illustrée sur la figure 4.12 pour une trace contenant une *flash crowd*. On peut voir que contrairement aux traces avec une attaque, la *flash crowd* n'engendre pas d'accroissement significatif de la distance. Ceci est cohérent avec les observations de la section précédente où les variations de α et β avec Δ ne permettaient pas de distinguer un trafic régulier d'un trafic avec *flash crowd*.

On évalue généralement la performance statistique d'une méthode de détection en traçant la courbe de probabilité de détection correcte P_D, en fonction de la probabilité de faux-positif P_F. Ce type de graphique se dénomme ROC (pour *Receiver Operationnal Characteristics*). En fonction du seuil fixé λ, on trace $P_D = f(\lambda)$ vs. $P_F = g(\lambda)$.

Nous avons obtenu ces courbes ROC expérimentalement avec notre répertoire d'attaques (présentées dans le tableau 4.3). Pour chaque seuil λ, nous avons déterminé les fenêtres dont la distance par rapport à la fenêtre de référence était supérieure à λ. Comme nous savons dans quelles fenêtres l'attaque a réellement eu lieu, nous pouvons estimer la probabilité de détection P_D (rapport, sur les fenêtres avec attaque, entre le nombre de fenêtres ayant une distance au

dessus du seuil et le nombre de fenêtres d'attaque) et la probabilité de faux-positifs P_F (rapport, sur les fenêtres sans attaque, entre le nombre de fenêtres ayant une distance au dessus du seuil et le nombre de fenêtres sans attaque). Si on note $\mathscr{F}$ l'ensemble des fenêtres de la trace, $\mathscr{F}_A$ l'ensemble des fenêtres où l'attaque a eu lieu, $\overline{\mathscr{F}_A}$ son complément, c'est à dire l'ensemble des fenêtres sans attaque, $\mathscr{F}_D$ l'ensemble des fenêtres dont le seuil dépasse λ, alors on a :

$$P_D = \frac{\text{card}(\mathscr{F}_D \cap \mathscr{F}_A)}{\text{card}(\mathscr{F}_A)}$$
$$P_F = \frac{\text{card}(\mathscr{F}_D \cap \overline{\mathscr{F}_A})}{\text{card}(\overline{\mathscr{F}_A})}$$

Pour les résultats présentés ici, la taille des fenêtres a été fixée à une minute et la fenêtre de référence correspondant à un morceau de dix minutes avant l'attaque. Δ est compris entre 2^1 et 2^{10} ms. Pour l'attaque Iperf-III, P_D vs. P_F ainsi que $P_D = f(\lambda)$ et $P_F = g(\lambda)$ sont présentées sur la figure 4.13. Sur une telle figure, le point idéal est le coin supérieur gauche (probabilité de détection égale à un et probabilité de faux-positifs nulle, donc toutes les attaques et uniquement les attaques sont détectées). Le pire cas est situé sur la diagonale, ce qui correspond à une détection au hasard (on choisit au hasard pour chaque fenêtre s'il s'agit d'une attaque ou non, la probabilité de détection est donc égale à la probabilité de faux positif). Les courbes montrent que la méthode d'estimation est assez performante. Nous avons aussi calculé, pour mettre mieux en évidence l'efficacité de la méthode, la probabilité de détection pour deux niveaux de faux positif fixé (10% et 20%). Ces valeurs sont reportées dans le tableau 4.4.

Anomalie	Génération	Id	Intensité (%)	P_D (10%)	P_D (20%)
fc	Humains	FC-1	31.27	04	14
fc	Humains	FC-2	18.35	19	25
DDoS	Iperf	R	33.82	91	93
DDoS	Iperf	I	17.06	51	64
DDoS	Iperf	II	14.83	48	54
DDoS	Iperf	III	21.51	48	58
DDoS	Iperf	IV	33.29	33	50
DDoS	Iperf	V	39.26	18	40
DDoS	Iperf	A	34.94	21	50
DDoS	Iperf	B	40.39	81	87
DDoS	Iperf	C	36.93	52	58
DDoS	Iperf	X	58.02	93	96
DDoS	Trinoo	tM	4.64	27	50
DDoS	Trinoo	tN	15.18	54	54
DDoS	Trinoo	tT	82.85	82	82

Tab. 4.4: Pour chaque attaque (chaque ligne), probabilités de détection étant donné un taux de faux positif fixé à 10% et 20%.

Les performances sont présentées ici pour des fenêtres d'une minute. Bien entendu, en augmentant la taille des fenêtres, on s'attend à avoir de meilleure performance, au détriment de la réactivité du système. Il serait intéressant, et cela fait partie des perspectives de recherche, d'ex-

plorer le compromis entre la taille des fenêtres et la performance de la méthode de détection.

Il est important de noter que c'est bien l'aspect multirésolution (évolutions $\alpha_\Delta, \beta_\Delta$) qui permet de détecter un trafic contenant une attaque d'un trafic légitime. Cela permet de s'affranchir des variations (légitimes) de débit dues au comportement variable des utilisateurs, et ainsi de détecter de manière efficace les attaques DDoS.

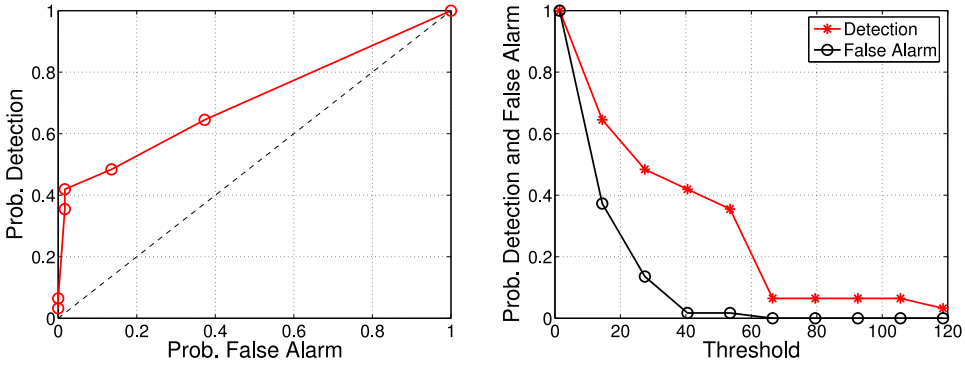


Fig. 4.13: Probabilité de détection P_D vs. probabilité de faux-positif P_F $P_D = f(P_F)$ (gauche), $P_D = f(\lambda)$ et $P_F = g(\lambda)$ pour $D_\alpha(l)$ (droite).

Les perspectives de recherches sont orientées maintenant vers la classification des anomalies à partir du modèle Gamma-FARIMA multirésolution introduit. Il faudra pour cela avoir accès à d'autres traces avec d'autres types d'attaques. Nous envisageons aussi de coupler cette méthode de détection avec des méthodes existantes (basées sur des profils de trafic par exemple) pour en améliorer les performances.

4.3 Simulation de trafic à longue mémoire

Cette section présente un travail concernant la simulation de trafic réseau, relativement indépendant des travaux présentés dans les chapitres précédents. Il contient deux grandes parties. Dans un premier temps nous avons cherché à valider expérimentalement une formule asymptotique pour générer du trafic à longue mémoire sur un réseau (section 4.3.3). Dans un second temps, afin de tester la méthodologie de détection d'attaques présentée dans la section 4.2, nous avons créé dans le simulateur de réseaux des attaques (section 4.3.4). La méthodologie suivie ainsi que le simulateur NS-2 sont décrits respectivement dans les sections 4.3.1 et 4.3.2. La publication de la première partie de ce travail est en cours, la seconde partie nous a permis de valider et de voir les limites de la méthode de détection proposée dans la section précédente.

Ce travail a été réalisé avec l'aide de deux stagiaires en projet de fin d'études au département Télécommunication Service & Usages de l'INSA de Lyon (Carole Montagnon et Fanny Jarlot).

4.3.1 Organisation et méthodologie

Le flot de simulation/analyse est présenté sur la figure 4.14. La première étape consiste à obtenir, grâce à un simulateur de réseaux (nous avons utilisé NS-2), des enregistrements de trafic correspondant à des scénarios précis (topologie, caractéristiques des sources, etc.). Les traces obtenues sont ensuite relues pour extraire des séries temporelles de débit agrégé à Δ_0. Ces séries font ensuite l'objet d'une analyse statistique à l'aide des méthodes et outils décrits dans les sections 2.7.3, 1.4.2 et 4.1.4.

4.3.2 Simulation de réseaux avec NS-2

Le simulateur NS-2 [112] est un simulateur de réseaux à événements discrets qui permet de simuler de nombreux scénarios sur définis par l'utilisateur. Le réseau est représenté (modélisé) par des sources de trafic (applications), des protocoles (UDP, TCP), des routeurs (avec leurs files d'attente) et des liens qui les relient. Le réseau est ensuite simulé, ce qui produit des traces et des statistiques.

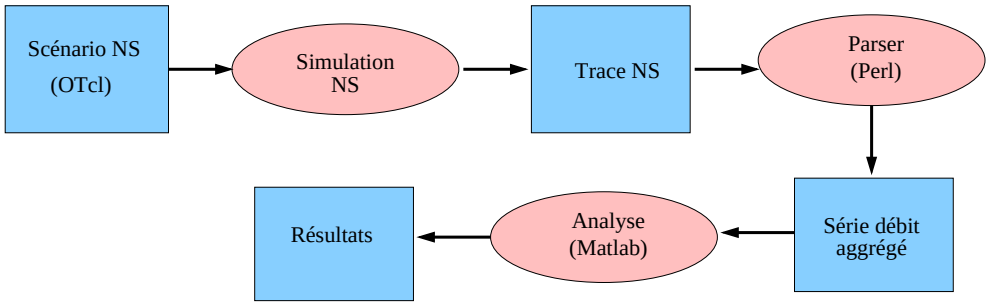


Fig. 4.14: Flot de simulation.

Les modèles de simulations de NS-2 sont écrits en C++ pour le fonctionnement interne des composants du réseau et en scripts OTCL, qui est la version objet du langage TCL[4], pour la description du réseau à simuler. Un modèle de réseau est constitué de différents modules :

[4] Tool Command Language

- **Nœuds**. Ils correspondent soit à des stations (génération de trafic) soit à des routeurs.
- **Liens** de communication. Ils modélisent les liens physiques entre deux nœuds.
- **Agents de communication**. Ils représentent les protocoles de niveau transport (TCP et UDP). Pour établir une communication entre deux nœuds, il faut attacher un agent sur chaque nœud, et les connecter pour que la communication (Flux UDP ou connections TCP) puisse avoir lieu.
- **Applications**. Elles sont en charge de la génération de trafic (transfert de fichier, trafic aléatoire, etc.) et utilisent les agents de communication.

Chaque module est largement paramétrable, on peut par exemple spécifier pour une application de transfert de fichier une distribution de la taille des fichiers, de la taille des paquets, etc..

En plus de la déclaration, du paramétrage et de l'interconnexion de tous les modules, on définit des événements dans le temps (démarrage/arrêt de la simulation, d'une application, etc.). On spécifie alors explicitement l'instant de l'événement. Voici une liste des principaux modules NS-2 que nous avons utilisés :

- **Générateur aléatoire** : une valeur numérique est tirée aléatoirement selon une loi statistique paramétrée. Nous avons essentiellement utilisé la loi uniforme pour la génération des débuts de connexions utilisateurs ou des tailles de paquets envoyés par exemple.
- **Agents de communications** : l'agent `Tcp` attaché à un nœud du réseau est lié explicitement dans le code à son nœud de destination, correspondant lui-même à un agent `TcpSink`. Les agents `Tcp` sont donc les émetteurs et doivent toujours avoir un agent `TcpSink` défini comme destinataire pour que l'échange de paquets TCP (connexion, accusés de réception, etc.) ait lieu. Les machines pirates présentes dans certains scénarios utilisent l'agent `Udp`. UDP étant un protocole qui, contrairement à TCP, ne nécessite pas de connexion. C'est pourquoi les nœuds de destination configurés pour recevoir le trafic UDP des pirates sont attachés à des agents "`Null`".
- **Applications** : nous avons paramétré trois types d'applications selon les scénarios vus plus loin. L'application `Pareto` est attribuée au mode d'émission des utilisateurs "normaux" du réseau. L'application `Cbr` (*Constant Bit Rate*) est rattachée au mode d'émission des machines "pirates". Enfin, l'application `Ftp` (transfert de fichiers) est affectée aux agents `Tcp` pour certains scénarios.
- **Programmateur** : ce module permet de gérer les débuts et fins de connexions de l'ensemble des nœuds du réseau modélisé. Il sert également à déterminer le temps total de la simulation.

4.3.3 Simulation de trafic à longue mémoire

L'hypothèse la plus communément admise quant à la présence de longue mémoire dans le trafic Internet est la distribution des tailles de fichiers (et donc des tailles de connexions) qui sont échangés sur le réseau Internet [78, 118, 41]. Cela a conduit au développement de sources de trafic de type ON/OFF (ON correspond au transfert d'un fichier, OFF correspond au temps qui s'écoule entre deux transferts de fichier), dans lequel la taille des temps ON (qui est proportionnelle à la taille du fichier à transférer) a une distribution à queue lourde (voir section 1.1.2). Taqu et Willinger ont en effet montré dans [156] que la superposition d'un grand nombre de sources de ce type avait pour conséquence l'émergence d'une caractéristique de longue mémoire de para-

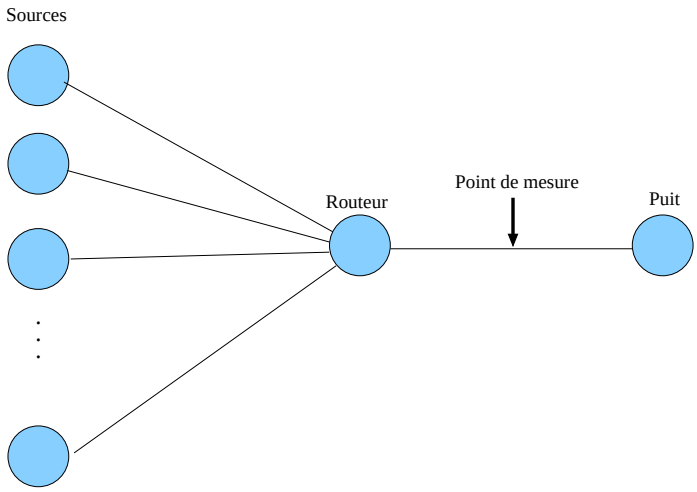


Fig. 4.15: Scénario pour retrouver la relation de Taqu-Williger.

mètre H directement lié à l'exposant de la queue de la distribution par la relation :

$$H = \frac{3-\alpha}{2}, \quad 1 < \alpha < 2 \tag{4.6}$$

Cette relation est doublement asymptotique, c'est à dire qu'elle est le résultat de la limite quand le nombre de source tend vers l'infini *et* de la limite quand le temps de simulation tend vers l'infini. Cette relation a été à la base de l'intégration dans les simulateurs de réseaux de source de trafic ON/OFF avec des temps ON distribués selon une loi de Pareto. La loi de Pareto possède en effet une queue qui se comporte en loi de puissance d'exposant α. On devrait donc, lorsque l'on simule un grand nombre de sources de ce type pendant une durée suffisamment grande, retrouver la relation (4.6). Ce type de source est d'ailleurs présent dans le simulateur NS-2, et il est même indiqué dans la documentation qu'il a été inclus dans le seul but de générer du trafic à longue mémoire. Certains travaux ayant effectué ce genre de simulation affirment que la relation de Taqu-Willinger est respectée [50, 116], cependant les résultats sont toujours assez écartés de la courbe théorique. Certains travaux exposent quant à eux une incapacité à retrouver la relation, affirmant que le paramètre de Hurst reste constant quand α varie [22]. Notre objectif a donc été d'éclaircir ce point. Notre premier objectif a donc consisté à vérifier la validité expérimentale de la relation $H = (3-\alpha)/2$.

Il est à noter que d'autres causes sont envisagées quant à la présence de longue mémoire comme celle décrite dans [53], nous avons néanmoins fait cette étude en ne considérant que la distribution des tailles de fichiers.

4.3.3.1 Scénario et paramètres

Afin de vérifier la validité de l'équation (4.6), nous avons construit le scénario classique illustré sur la figure 4.15. Il est composé de N sources émettant toutes un trafic de même nature vers un puit. Toutes les communications passent par un routeur, et nous analysons le trafic sur le lien entre le puit et le routeur.

Ce scénario est caractérisé par une série de paramètres. A chaque configuration (scénario et valeurs des paramètres fixés) correspond une simulation différente. Voici les paramètres NS-2 pris en compte ainsi que la plage de valeurs que nous avons utilisés :

- T : temps total simulé. Nous avons toujours effectué des simulations de deux heures donc la valeur attribuée à ce paramètre a été fixée à 7200s.
- N : nombre de machines émettrices (sources). Valeurs choisies : {5 ; 20 ; 50 ; 100}.
- D_{on} : débit d'émission des paquets distribués selon une loi de Pareto (donc pendant les temps "on" des sources émettrices). Valeurs testées : { 500 kb/s, 1 Mb/s, 2Mb/s, 3Mb/s }
- t_{on} : durée moyenne pendant laquelle une source considérée va émettre ses paquets. Valeurs simulées : {1s, 100ms, 10ms, 5ms}. Cela correspond au paramètre d'échelle de la loi de Pareto.
- t_{off} : durée moyenne pendant laquelle une source considérée n'émettra rien sur le réseau. Valeurs choisies : {1s, 100ms, 10ms, 5ms }.
- α : paramètre de forme de la loi de Pareto (exposant de la queue de la distribution relié à H par la relation (4.6). Nous avons pris comme plage de valeurs : {1,2 ; 1,4 ; 1,6 ; 1,8 ; 2 ; 2,2 ; 2,4 ; 2,6 ; 3 ; 4}.
- S : Taille des paquets envoyés par les sources. Afin de se rapprocher de la réalité d'un trafic utilisateur, nous avons généré aléatoirement la taille des paquets émis selon une loi Uniforme de paramètres entre 200 et 1000 qui correspondent respectivement à la taille minimale et maximale que peuvent avoir les paquets (en bits).
- Nous avons également testé la génération aléatoire des paramètres suivants : t_{on} et t_{off} (suivant une loi Uniforme de paramètres (5, 100) représentant une gamme de valeurs en millisecondes) ; D_{on} (suivant une loi Uniforme de paramètres (500, 3000) représentant une gamme de valeurs en Kb/s).

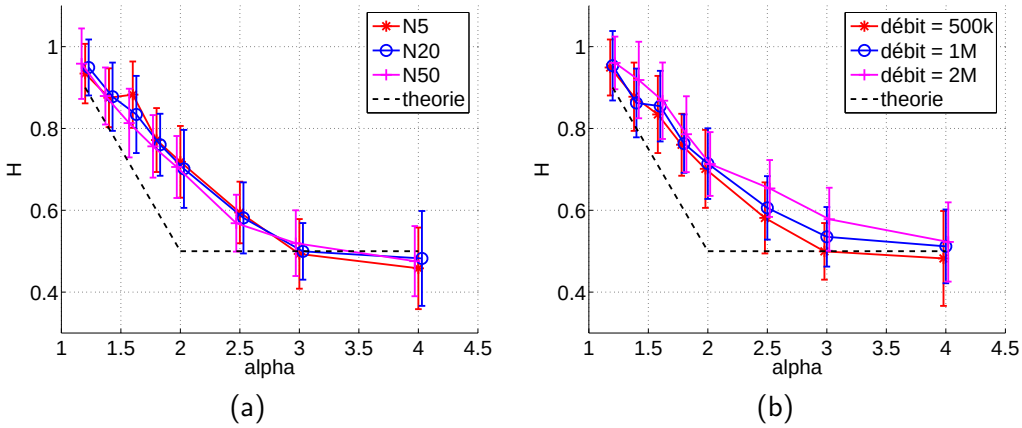
Pour chaque simulation, le flot présenté sur la figure 4.14 a été suivi, et l'estimation du paramètre de longue mémoire H a été effectué par l'estimateur en ondelette présenté dans la section 2.7.3. Nous avons, à partir des résultats obtenus, tracé des courbe $H = f(\alpha)$ afin de vérifier la validité de l'équation (4.6).

4.3.3.2 Résultats

Après avoir exploré de nombreuses configurations, il est finalement apparut que la relation entre H et α pouvait être vérifiée expérimentalement, mais à condition de prendre garde à certains paramètres. Parmi tous ceux que nous avons explorés, il est apparu que ce sont les temps moyens d'émission et d'arrêt (t_{on} et t_{off}) qui permette d'obtenir les meilleurs résultats. Ce temps moyen correspond au paramètre d'échelle de la loi de Pareto. Nous décrivons ci-après les résultats sur chaque paramètre.

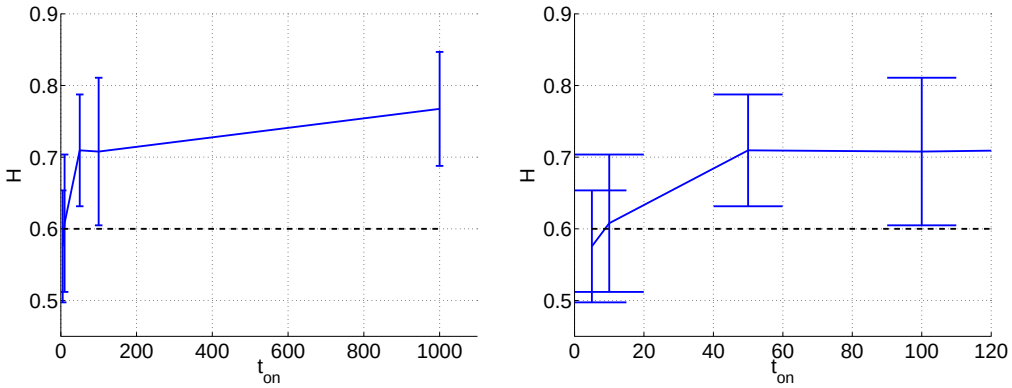
• **Nombre de sources** N. La figure 4.16(a) montre des courbes de H en fonction de α pour différents nombres de sources. Les paramètres t_{on} et t_{off} sont ici fixés à 1s. Les courbes expérimentales sont non seulement très éloignées de la relation théorique (en pointillé), mais aucune amélioration sensible n'est observée quand le nombre de sources augmente. Il semble donc d'une part que le nombre de sources n'a pas un rôle déterminant pour retrouver la relation théorique, et d'autre part que cette relation n'est pas valable pour les paramètres de ces simulations.

• **Débit d'émission** D_{on}. La figure 4.16(b) montre des courbes $H = f(\alpha)$ pour différents débits d'émission. Les paramètres t_{on} et t_{off} sont ici fixés à 1s. Les courbes expérimentales sont encore très loin de la relation théorique, on peut observer que de faibles débits semblent montrer de meilleurs résultats, mais sans que cela soit satisfaisant.



(a) (b)

Fig. 4.16: Relation $H = f(\alpha)$ pour différents nombre de sources (a) et débits (b).

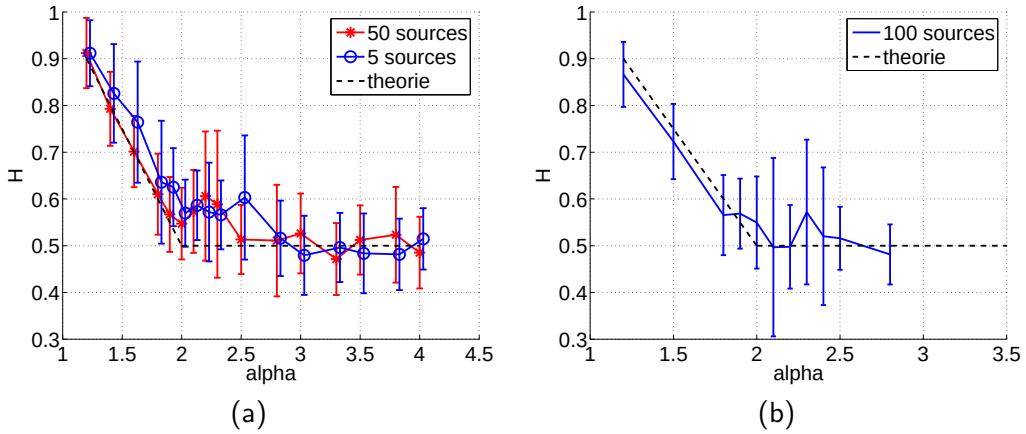


(a) (b)

Fig. 4.17: Relation $H = f(\alpha)$ pour différents t_{on} (a) et un zoom pour les petits t_{on} (b).

• **Temps t_{on} et t_{off}.** Dans toutes les simulations, nous avons gardé $t_{on} = t_{off}$. La figure 4.17 montre l'évolution de l'estimation de H à α fixé ($\alpha = 1.8$) en fonction de t_{on}. Il est clairement mis en évidence que l'estimation est plus proche de la valeur théorique lorsque t_{on} est petit. La figure 4.18(a) montre que cette fois, avec des t_{on} petits, on arrive à retrouver la relation recherchée. La figure 4.18(b) montre la meilleure courbe que nous avons obtenue avec un grand nombre de sources. On peut donc affirmer que pour simuler un trafic à longue mémoire de paramètre H donné en utilisant des sources de Pareto, alors il faut prendre garde à fixer un t_{on} et un t_{off} assez faibles (de l'ordre de 10 ms).

Ce résultat peut être interprété de la manière suivante. La relation (4.6) étant asymptotique en temps, le diagramme LD devrait ressembler à celui présenté figure 4.19. Pour être cohérente avec la théorie, la valeur de H doit être estimée dans la limite des grandes échelles. La figure 4.19 illustre bien le problème auquel on est confronté : la courbe ne tendant vers son asymptote que très lentement, on croit faire une régression linéaire valide alors qu'on la fait trop tôt car la simulation n'est pas assez longue. Comme présenté sur la figure, cela revient à faire une sur-estimation de H, ce qui est cohérent avec les courbes de la figure 4.16. Nous ne pouvions pas augmenter le temps simulé car les simulations étaient déjà très longues et prenaient beaucoup de mémoire. En effet, une simulation de deux heures avec 20 sources prend environ 8h sur un centrino à 1,6 Ghz/1Go de RAM, et la trace récupérée en sortie fait entre 1 et 2 Go. Le temps comme la taille de la trace augmente linéairement avec le nombre de sources.



(a) (b)

Fig. 4.18: Relation $H = f(\alpha)$ pour t_{on} petit (5 ms)et différents nombres de sources (a) et pour 100 sources (b).

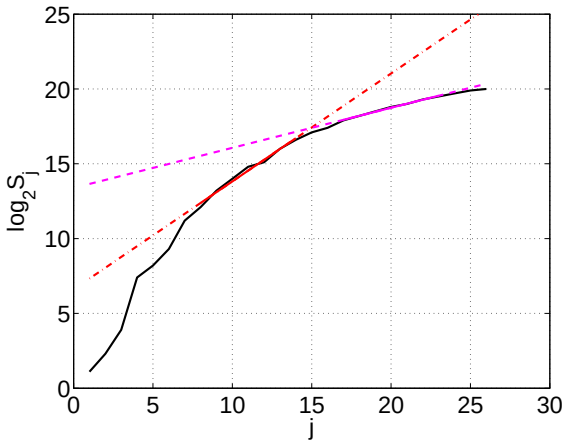


Fig. 4.19: Vue (d'artiste) du diagramme LD sur une large gamme d'échelles, illustration de l'erreur d'estimation de H lorsque la simulation est trop courte.

Néanmoins, en jouant sur le paramètre t_{on}, on *décale* artificiellement le diagramme vers les petites échelles (vers la gauche), et en faisant la régression linéaire sur la même gamme d'échelles, on obtient une estimation de H bien plus proche que celle théoriquement attendue car la courbe est alors bien plus proche de son asymptote.

Ce résultat a été confirmé par des simulations MATLAB de sommes de processus ON/OFF avec des distributions de Pareto pour la taille des temps ON et OFF menée par Pierre Borgnat (Laboratoire de Physique, ENS Lyon). En effet, le résultat de ces simulations est que pour obtenir l'équation (4.6) en un temps de simulation raisonnable, alors il faut fixer la valeur moyenne des temps ON et OFF a une valeur faible devant l'échelle à partir de laquelle la longue mémoire est prédominante (expérimentalement de l'ordre de 1s). Ces résultats feront prochainement l'objet d'une publication.

4.3.4 Détection d'attaques

Dans cette section, nous présentons la mise à l'épreuve de la méthode de détection présentée dans la section 4.2 par des simulations toujours dans NS-2. Nous avons pour cela mis en place

différents scénarios d'attaques. Nous avons surtout exploité les résultats sur une attaque de vol de bande passante. En effet, notre objectif était de détecter des attaques de petites tailles (qui ne provoquent pas d'augmentation forte du débit). En effet les attaques de grandes tailles sont facilement détectables en observant l'évolution de la moyenne et/ou la variance du débit agrégé.

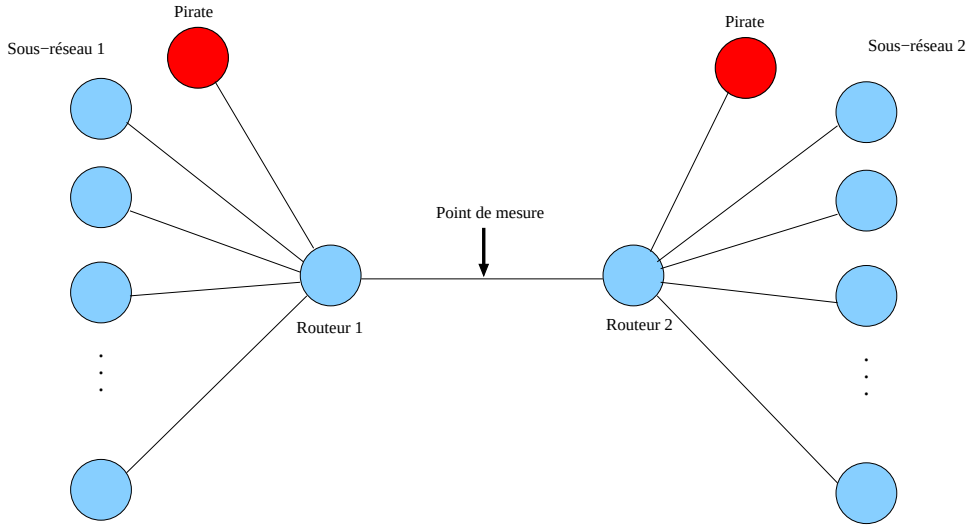


Fig. 4.20: Scénario d'attaque de bande passante.

Le scénario que nous avons mis en place est présenté sur la figure 4.20. Les sources normales effectuent des transferts de fichiers en utilisant le protocole FTP, et la distribution de la taille des fichier est une loi de Pareto afin de reproduire la caractéristique classique de longue mémoire dans le trafic. Les communications ont seulement lieu entre le sous-réseau 1 et le sous-réseau 2. Les pirates quant à eux émettent, pendant les périodes d'attaques, un flux `Cbr` (*Constant Bit Rate*) de faible intensité. Afin de pouvoir tester la détection, nous avons fixé des périodes d'attaques comme indiqué sur la figure 4.21.

Le même flot de simulation (figure 4.14) a été suivi mais cette fois les analyses statistiques consistent en le calcul des paramètres du modèle multirésolution introduit dans la section 4.2. Nous avons expérimentalement vérifié que la loi Gamma était une bonne candidate pour la modélisation des traces de débit agrégé, et ceci est illustré par la figure 4.22, qui montre deux histogrammes empiriques (avant et pendant une attaque) ainsi que l'ajustement par une loi Gamma, pour deux niveaux d'agrégation. L'ajustement est, comme cela a déjà été montré précédemment, tout à fait satisfaisant.

Nous avons ensuite calculé les évolutions $\hat{\alpha} = f(\Delta)$ et $\hat{\beta} = f(\Delta)$ comme expliqué dans la section 4.2. Dans un premier temps, le plus petit niveau d'agrégation a été fixé à 10 ms. Les courbes

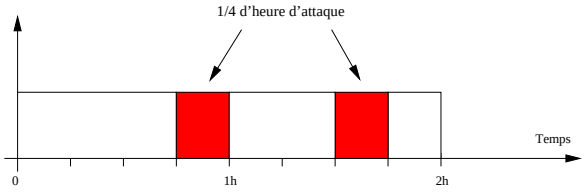


Fig. 4.21: Périodes d'attaques pour le scénario de vol de bande passante.

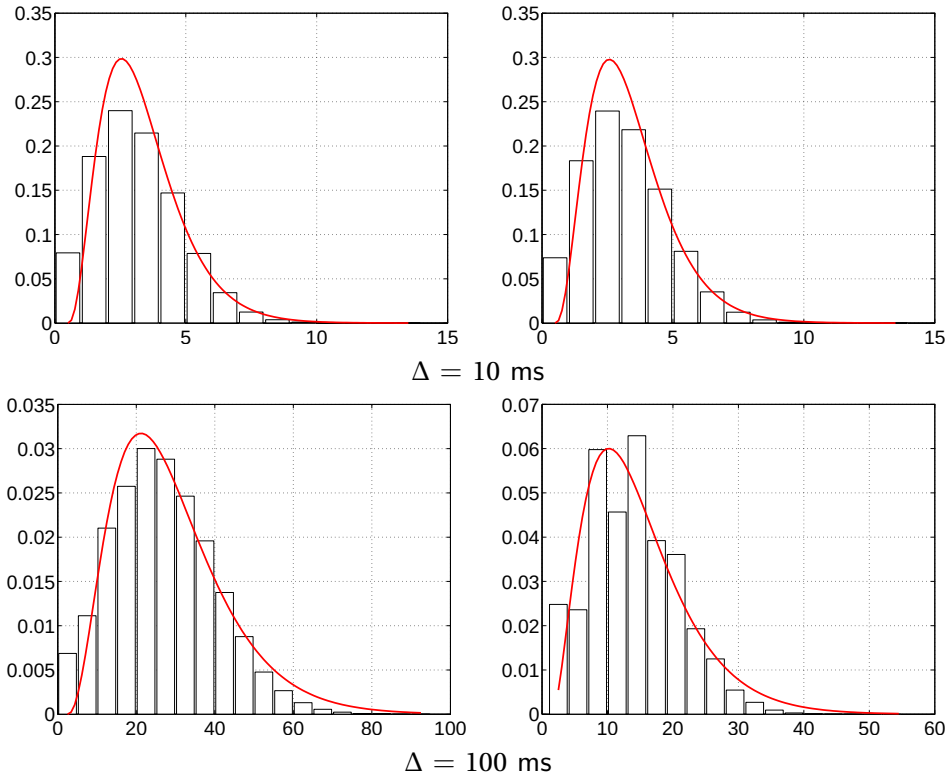


Fig. 4.22: Loi marginales de deux séries de débit agrégé à 10 ms, une avant l'attaque (colonne de gauche) et une pendant l'attaque (colonne de droite).

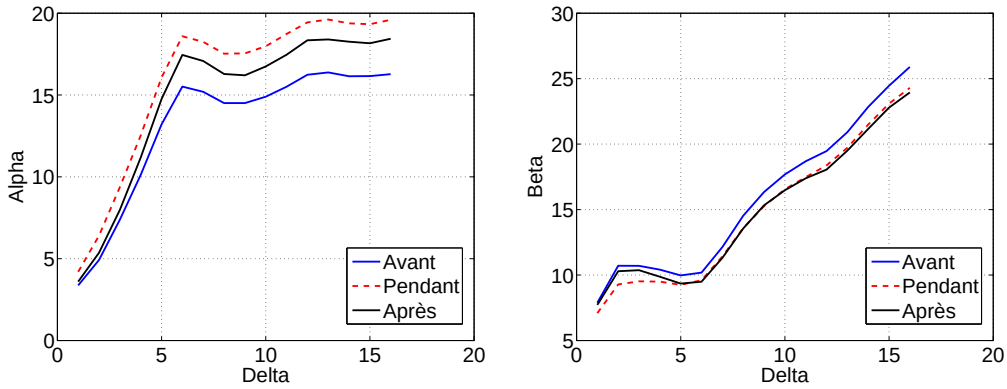


Fig. 4.23: Évolution des paramètres α (gauche) et β (droite) de la loi Gamma en fonction deΔ (agrégation minimum $\Delta_0 = 10$ ms), pour chaque quart d'heure de la trace. Les quarts d'heure d'attaque sont tracés avec une plus grande épaisseur.

de la figure 4.23 montrent ces évolutions pour trois quarts d'heure de la trace (avant, pendant et après une attaque). On voit bien que l'on ne peut pas, sur cette figure, distinguer les périodes d'attaques des périodes régulières. En effet les courbes ont une forme similaire.

Nous avons donc décidé de regarder les évolutions $\hat{\alpha} = f(\Delta)$ et $\hat{\beta} = f(\Delta)$ pour Δ plus petit, nous avons donc utilisé des données agrégées à $\Delta_0 = 1$ ms comme base. Ces évolutions sont illustrées sur la figure 4.24, et dans ce cas on peut observer un comportement différent pour les périodes d'attaques et les périodes régulières. En effet, on observe une différence marquée entre

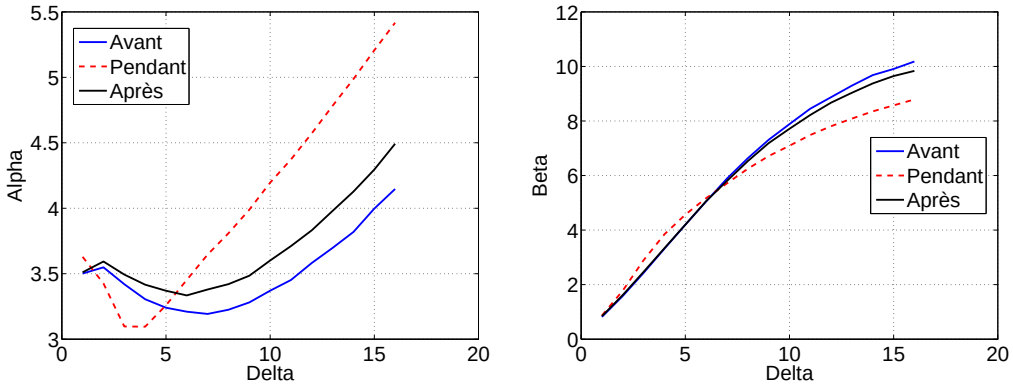


Fig. 4.24: Évolution des paramètres α (gauche) et β (droite) de la loi Gamma en fonction de Δ (agrégation minimum $\Delta_0 = 1$ ms), pour trois quarts d'heure différents de la trace (avant, pendant et après l'attaque).

la forme des courbes pendant l'attaque et en dehors de l'attaque.

On peut donc en conclure que la méthode détection que nous avons proposé fonctionne, à condition que le niveau d'agrégation initial Δ_0 soit petit (de l'ordre d'1 ms). Il est à noter que ceci est cohérent avec les résultats de la section 4.2.6 qui évalue les performances de la méthode de détection sur des traces réelles. Ces résultats ont été obtenus sur des bouts de trace d'un quart d'heure (900000 ms !). Cela n'est pas envisageable en pratique, il faut que la méthode soit plus réactive, c'est à dire qu'elle soit capable de détecter des attaques en regardant des morceaux plus petits.

4.4 Conclusion

Les expérimentations que nous avons effectuées avec le simulateur de réseaux NS-2 nous ont permis d'une part d'explorer la validité expérimentale de la relation de Taqu-Willinger utile pour la génération de trafic à longue mémoire, et d'autre part de mettre à l'épreuve notre méthode de détection d'anomalies présentée en détail dans la section 4.2. La simulation permet de contrôler de manière très précise les différents paramètres du réseau, et offre donc un outil important pour évaluer une méthode de détection d'anomalie. Ces simulations nous ont permis de mettre le doigt sur une limite de notre méthode de détection, ce qui nous a permis d'envisager de manière plus claire les perspectives de recherches dans cette voie qui seront détaillées dans la conclusion de cette première partie.

Conclusion et perspectives

Nous avons présenté, dans cette première partie, le fruit de notre collaboration avec le laboratoire de Physique de l'ENS de Lyon (Équipe SISYPHE). Nous avons exploré le problème de la synthèse de réalisations de processus à longue mémoire non-gaussiens, ainsi que la problématique de la modélisation de trafic Internet, en particulier dans le but de détecter des anomalies.

Notre contribution a plus été orientée sur la modélisation de trafic, l'analyse et traitement de large volume de données, que sur l'aspect théorique du problème, qui reste encore ouvert dans le sens où il n'existe pas de méthode pour générer un processus ayant une covariance *et* une loi marginale données arbitraires. Nous avons néanmoins proposé et implémenté une méthode de synthèse de processus à longue mémoire non-gaussien [SA05, SLB$^+$06a, SLO$^+$05], qui est à l'heure actuelle utilisée par le LAAS pour générer du trafic dans des simulateurs de réseaux. Cette méthode est aussi utilisée dans les travaux de la deuxième partie concernant la génération de trafic sur puce. Un tel générateur est un outil très intéressant pour la métrologie et l'analyse de trafic Internet. Les processus non gaussiens à longue mémoire n'étant pas seulement présent dans les traces de trafic, mais aussi dans divers domaine (turbulence, imagerie médicale, etc.), l'outil que nous avons développé devrait être utile à d'autres communautés scientifiques.

Nous avons d'autre part proposé une modélisation de trace de débit agrégé par le modèle GAMMA – FARIMA). Nous avons montré que ce modèle capture de manière très satisfaisant les caractéristiques statistiques d'ordre un (marginales) et deux (covariance), et ce pour une large gamme de niveau d'agrégation.

Ce constat nous a permis de proposer un modèle *multirésolution* de trafic, et nous a permis de mettre au point une méthode de détection d'anomalies. C'est en effet en observant l'évolution des paramètres du modèle avec le niveau d'agrégation Δ (d'où la nature multirésolution de la modélisation), que nous avons pu distinguer un trafic régulier, d'un trafic incluant une attaque de déni de service distribué [BLO$^+$06]. L'ensemble de ces travaux (synthèse de processus à longue mémoire non-gaussiens, modélisation de trafic Internet et détection d'anomalies) a été soumis dans un article de synthèse au journal *IEEE Transaction on dependable Computing*.

Enfin, nous avons utilisé un simulateur de réseau (NS-2) pour étudier la génération de trafic à longue mémoire d'un point de vue purement expérimental. Nous avons en effet cherché à reproduire expérimentalement une analyse théorique de Taqu et Willinger qui stipule qu'une superposition d'un nombre infini de source on/off avec des périodes "on" distribuées selon une loi à queue lourde introduit dans le trafic une longue mémoire dont le paramètre est relié à l'exposant de la queue de la distribution. Nous avons cherché (et réussi) à trouver les conditions expérimentales permettant de reproduire cette loi de manière satisfaisante, ce qui n'était pas le cas dans la littérature. Nous avons aussi, par simulation dans NS-2, mis à l'épreuve notre technique de détection d'anomalies sur des scénarios simples. Les résultats de cette étude tendent à montrer que cette méthode, si elle permet bien d'identifier une attaque, reste difficile à implémenter réellement car il est nécessaire d'avoir des enregistrements de trafic faiblement agrégé, et de les enregistrer sur une durée non-négligeable (plusieurs minutes). Nous avons aussi utilisé le simulateur NS-2 pour mettre à l'épreuve notre technique de détection d'anomalie sur des scé-

narios simples. Les résultats de cette étude tendent à montrer que cette méthode, si elle permet bien d'identifier une attaque, reste difficile à implémenter réellement car il est nécessaire d'avoir des enregistrements de trafic faiblement agrégé ($\Delta = 1$ ms), et de les enregistrer sur une durée non-négligeable (plusieurs minutes).

Les perspectives de recherches incluent une étude complémentaire pour évaluer la faisabilité de l'intégration de cette méthode de détection dans un outil de supervision, ainsi que l'intégration de modèles de loi marginale et de covariance supplémentaire dans notre outil de génération de processus à longue mémoire non-gaussien. D'une manière générale, l'utilisation de modélisation statistiques avancé dans le domaine de la métrologie est un domaine relativement récent où nombre d'aller retour entre théorie (traitement du signal, processus stochastiques, etc.) et pratique (enregistrement de données, réalisation d'attaque, génération de trafic, simulation, etc.) restent encore à explorer. Nous pensons que l'utilisation de de telles modélisations dans ce domaine constitue une voie de recherche riche et intéressante tant dans le domaine de l'évaluation des réseaux de machine, que dans le domaine de l'évaluation des réseaux sur puce, qui est l'objet de la seconde partie de ce manuscrit.

Deuxième partie

Étude des communications sur puce

Introduction

Cette partie concerne la modélisation et la génération de trafic résultant des communications entre les différents composants intégrés au sein d'une même puce de silicium (système sur puce, SoC pour *Systems on chip*). Ces systèmes entrent aussi dans la catégorie des systèmes embarqués (*embedded systems*) puisqu'ils sont utilisés dans le cadre d'applications embarquées (automobile, aviation, téléphonie, électronique grand public, etc.).

Ce sujet de recherche a pris ces dernières années une importance croissance, car le composant en charge des communications au sein d'une puce est en train de vivre un changement technologique majeur avec la transition des systèmes à base de bus partagés vers des petits réseaux baptisés *réseaux sur puce* (NoC pour *Network on chip*). Ces nouvelles interconnexions sont encore en cours d'étude et afin de pouvoir évaluer leurs performances, les comparer entres elles et guider les choix de leur conception et il faut pour cela observer leur comportement sous une charge de trafic réaliste. La simulation des systèmes sur puce prenant un temps considérable lorsque les composants sont simulés de manière précise, on utilise, pour effectuer le dimensionnement du réseau sur puce, des générateurs de trafic qui sont très simples et donc rapides à simuler. C'est sur la problématique de la conception de ces générateurs de trafic qu'est centrée cette partie de la thèse. Plus précisément, nous avons proposé un environnement de génération de trafic adapté à l'évaluation de performance des réseaux sur puce.

Il existe différents types de générateurs de trafic. On peut utiliser, dans les premiers temps du développement (pour faire des choix à grande échelle), des générateurs produisant une charge aléatoire. On peut alors par exemple tracer l'évolution de la latence moyenne du réseau en fonction de la charge moyenne injectée et ainsi comparer différentes topologies, algorithmes de routage, etc., comme cela a été fait il y a déjà quelques années pour les réseaux de machines, et comme cela est encore étudié pour les réseaux hautes performances (comme les grilles de calculs par exemple). La conception de réseaux sur puce doit d'ailleurs tirer parti de ces recherches, cependant les NoC disposent de contraintes spécifiques qu'il ne faut pas oublier de prendre en compte. Lorsque l'application qui va s'exécuter sur la puce commence à être mieux définie, il faut dimensionner le réseau pour cette application. On pourra à cette étape utiliser des générateurs de trafic écrits par les concepteurs des différents composants de la puce (à partir d'une connaissance précise du comportement de chaque composant), ou envisager une approche de rejeu, c'est à dire rejouer des traces de communications enregistrées lors d'une simulation. On pourra aussi, comme nous allons le montrer dans cette partie, modéliser le trafic à l'aide de processus stochastiques afin d'en capturer les caractéristiques statistiques et utiliser ces modèles pour générer du trafic réaliste dans les SoC.

Notre approche se place entre l'utilisation d'une charge aléatoire et l'utilisation de générateurs de trafic précis écrits par les concepteurs des composants, et cette partie s'efforcera de montrer d'une part qu'une modélisation stochastique du trafic sur puce n'a de sens que si on tient compte des différentes *phases* du trafic, et d'autre part que cette approche représente un bon compromis entre précision et temps de simulation, et donc que notre environnement de génération de trafic est utile.

Cette partie est organisée comme suit : le chapitre 5 présente des généralités sur les SoC ainsi qu'un état de l'art centré sur la problématique de la génération de trafic dans les SoC. Le chapitre 6 présente notre approche de la génération de trafic, le générateur de trafic que nous avons développé ainsi que le flot d'évaluation de performance de réseaux sur puce mis en place, et le chapitre 7 décrit l'environnement de simulation que nous avons utilisé dans nos expérimentations, ainsi que les développements logiciels que nous avons effectués dans cet environnement. Enfin le chapitre 8 présente des résultats expérimentaux justifiant la validité de notre approche et une conclusion discutera les contributions ainsi que les perspectives de recherche.

Chapitre 5
État de l'art

Ce chapitre rassemble une revue des travaux existants concernant les interconnexions sur puce et la génération de trafic sur puce. Il est organisé comme suit : la première section reporte des généralités sur la conception de systèmes sur puce (section 5.1), la section 5.2 contient ensuite un état de l'art sur les réseaux sur puce et une présentation détaillée des communications ayant lieu au sein d'une puce est proposée dans la section 5.3. Enfin la section 5.4 décrit les travaux récents qui ont le même objectif que celui poursuivi dans cette partie, à savoir la génération de trafic dans les systèmes sur puce.

5.1 Systèmes sur puce

Cette section définit dans un premier temps ce qu'est un système sur puce dans la section 5.1.1 et introduit la problématique de la conception de ces systèmes dans la section 5.1.4.

5.1.1 Qu'est qu'un SoC

Un SoC correspond à l'assemblage, au sein d'une même puce de silicium, de différents composants comme cela est illustré sur la figure 5.1, qui montre un schéma générique d'un SoC et sur la figure 5.2 qui montre le SoC inclus dans un assistant personnel Hewlet-Packard (IPAQ modèle H5500). L'évolution rapide de la quantité de transistors pouvant être intégrés dans un circuit électronique (suivant la loi de Moore, c'est à dire que ce nombre double tous les 18 mois) permet de réaliser des circuits de plus en plus complexes. On peut maintenant se permettre d'intégrer, dans un même circuit, un grand nombre de composants.

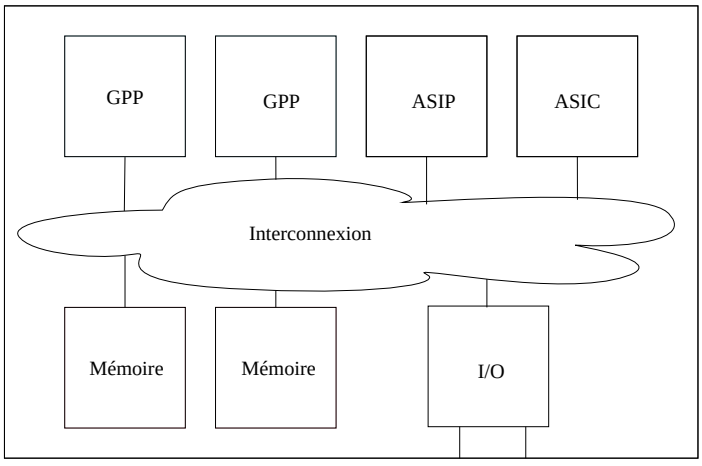


Fig. 5.1: Schéma bloc d'un système sur puce standard.

L'ensemble des composants est *fondu* par un procédé qui consiste à les convertir en un gigantesque circuit électronique uniquement formé de transistors (implantés dans le silicium), interconnectés par des pistes métalliques insérées sur plusieurs niveaux (pour arriver à tout interconnecter). On obtient alors ce que l'on appelle une puce, c'est à dire un morceau de silicium de quelques millimètres carrés incluant plusieurs centaines de millions de transistors interconnectés. Le niveau de miniaturisation a atteint un niveau tel qu'il faudrait la surface d'un terrain de football pour avoir un plan lisible à l'œil d'une puce. Ce procédé de fabrication est très complexe et implique l'utilisation de différents jeux de masques (comme pour l'impression des circuits imprimés), dont le prix est exorbitant (environ 1 million de dollar). Ceci a pour conséquence que l'on ne peut pas faire de petites séries de systèmes sur puce, il faut les produire en masse pour qu'ils soient rentables.

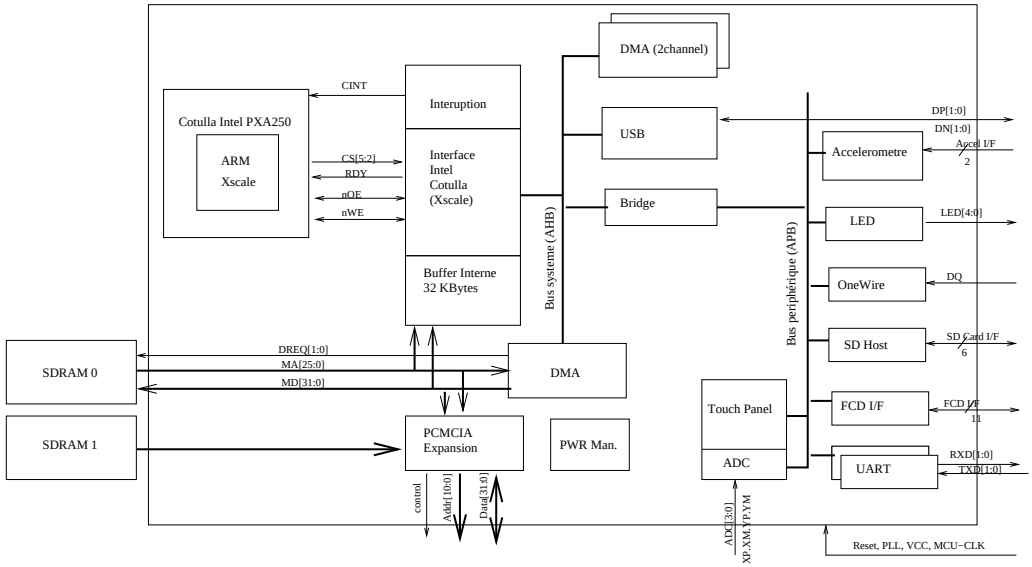


Fig. 5.2: Schéma du système sur puce utilisé dans l'assistant personnel IPAQ HP5500.

Les systèmes sur puce sont utilisés lorsqu'il y a un besoin fort d'intégration (les différents composants pourraient dans le cas contraire être placés sur une carte, ce qui est beaucoup moins coûteux et moins complexe à réaliser) et/ou de calculs intensifs (les composants d'un SoC sont très proches, donc ils peuvent communiquer plus rapidement). On peut citer, comme applications typiques des SoC, les téléphones portables, les assistants personnels (PDA pour *Personal Digital Assistant*), les *set-top box* (boîtier multi-fonction pour le son et la vidéo, décodage de télévision satellite, DVD et DivX, enregistrement, arrêt sur image, lecture en différée, etc.), les consoles de jeux portables comme la PSP (pour *Playstation* portable) par exemple. Les SoC appartiennent donc à la famille plus vaste des *systèmes de calculs embarqués* (appelés plus communément *systèmes embarqués*) qui couvrent l'ensemble des systèmes numériques complexes qui évoluent dans des milieux mobiles (téléphonie, ordinateur de poche, électronique embarquée dans une automobile).

La logique utilisée dans la plupart des SoC est une logique dite *synchrone*, ce qui implique l'existence d'un signal d'horloge qui synchronise tous les éléments de la puce (la fréquence de cette horloge est la fréquence de fonctionnement du SoC). L'intégration est maintenant telle que la propagation de ce signal d'horloge dans la puce est problématique (aux fréquences actuelles, le signal met physiquement plusieurs cycles d'horloge pour aller d'un bout à l'autre de la puce). On va donc vers des SoC localement synchrones (les composants sont constitués de logique syn-

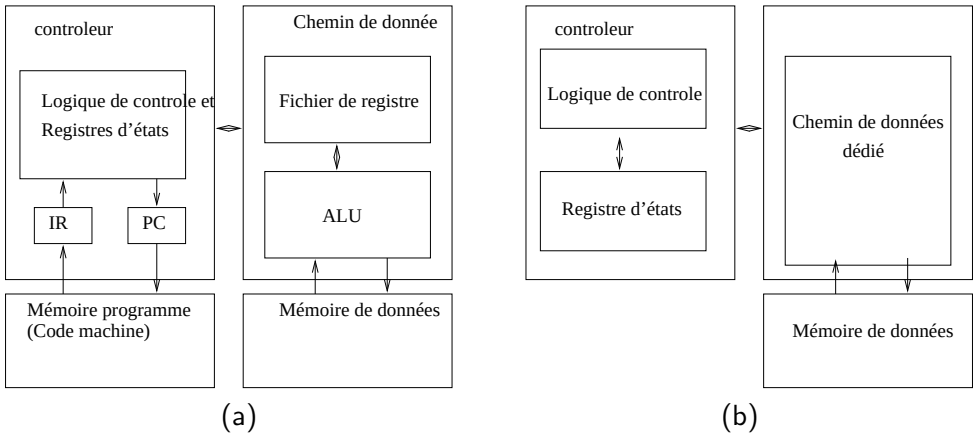


Fig. 5.3: Schéma de fonctionnement des différents type de composants de calculs : les processeurs (généraliste et spécialisée) (a) et les composants dédiés (b).

chrone, il dispose donc d'une horloge) et globalement asynchrones car les différentes horloges des composants ne seront pas issues d'un même signal et pourront potentiellement fonctionner à des fréquences différentes. On parle alors de conception GALS (*Globally Asynchronous, Locally Synchronous*). Il existe aussi des puces *asynchrones*, qui sont peu consommatrices, mais le manque d'outil performant pour aider à leur conception fait que cela reste une activité marginale dans le domaine de la micro-électronique.

5.1.2 Composants de calculs

Voici une description des différents types de composants de calculs qui sont intégrés dans ces systèmes :

- **Processeurs généralistes** (GPP pour *General Purpose Processor*). Ce sont des processeurs "classiques", ils sont capables d'exécuter du code binaire (un flot d'instructions) issu de la compilation d'un code assembleur, C, C++, etc. Deux espaces mémoires distincts sont utilisés pour le code et pour les données. Le temps d'accès à la mémoire (dizaines ou centaines de cycles d'horloge) étant très long devant le temps de calcul (quelques cycles), ces processeurs sont en général couplés à une mémoire cache, dont le fonctionnement sera détaillé dans la section 5.3.3.2. Le fonctionnement de base d'un processeur est décrit sur la figure 5.3(a), où l'on peut distinguer les deux espaces de stockages différents pour le code et les données, l'un servant à fournir le flot d'instructions et l'autre les opérandes de ces instructions. Les deux registres d'états correspondent à l'adresse de l'instruction en cours (PC pour *Program Counter*) et à l'instruction en cours (IR pour *Instruction Register*). Il existe de nombreux types de processeurs, et l'objet de ce travail n'en nécessite pas une liste détaillée. On peut par exemple distinguer les processeurs ayant un *jeu d'instruction* (ensemble des instructions que le processeur peut exécuter) simple (RISC pour *Reduced Instruction Set Computer*, quelques dizaines d'instructions correspondant à des opérations simples) ou complexe (CISC pour *Complex Instruction Set Computer*, quelques centaines d'instructions correspondant souvent à plusieurs opérations). Dans les SoC, on utilise plutôt des processeurs RISC qui ont une architecture plus simple que les CISC. Sur le schéma de l'IPAQ (figure 5.2), il y a un seul processeur généraliste (ARM *Xscale*) sur la gauche. C'est lui qui va se charger de piloter tous les autres composants.
- **Processeurs spécialisés** (ASIP pour *Application Specific Instruction-set Processor*). Ce sont

des processeurs qui ont été optimisés pour certains traitements, typiquement des traitements réguliers associés à du traitement de signal (on parle alors de DSP pour *Digital Signal Processor*). Ils disposent d'instructions spéciales pour faire certaines fonctions (la transformée de Fourier rapide ou la convolution par exemple) et ils disposent de plusieurs accès à la mémoire en parallèle. Le schéma de fonctionnement est cependant identique au GPP (figure 5.3(a)).

- **Accélérateurs matériels** (ASIC pour *Application Specific Integrated Circuit*). Ce sont des circuits spécialisés pour réaliser une fonction donnée. Contrairement aux processeurs, il n'y a pas dans ce cas de code à exécuter, l'algorithme est câblé physiquement par un assemblage de portes logiques. Ils sont en général utilisés pour effectuer des calculs intensifs très rapidement (l'estimation de mouvement dans MPEG2 par exemple). Le schéma de fonctionnement de tels composants est représenté sur la figure 5.3(b). On note qu'il n'y a plus de code, mais simplement un accès dans la mémoire de données. Afin d'accélérer les accès mémoire de ces composants, ils disposent souvent d'un DMA (*Direct Memory Access*), ils peuvent ainsi effectuer des accès en rafales à la mémoire sans passer par un processeur et être ainsi très performants.
- **Périphériques**. Ce sont de petits composants dédiés à une tâche précise concernant le pilotage d'un élément extérieur à la puce (gestion du port USB, de l'interface réseau, d'un affichage digital, etc.). Ils sont par exemple rassemblés à droite sur la figure 5.2.

Il existe encore de nombreuses déclinaisons qui pourraient s'insérer entre les quatre grandes catégories présentées ici et on pourrait tracer une courbe de compromis flexibilité/performance. En effet, les processeurs généralistes sont très flexibles puisqu'il peuvent exécuter n'importe quel algorithme écrit dans un langage de haut niveau, en revanche ils sont peu performants (pour effectuer un calcul, il faut aller chercher l'instruction dans la mémoire, la décoder, effectuer le calcul et renvoyer le résultat dans la mémoire). Les ASIC au contraire sont très rapides, en revanche ils ne sont pas programmables. Les DSP constituent un compromis entre les deux. On pourrait de même tracer une courbe flexibilité/consommation, montrant que plus un composant de calcul est spécialisé, moins il consomme d'énergie.

Afin que tous les composants puissent communiquer entre eux, il faut un composant d'interconnexion en charge des communications. Ce composant fera l'objet d'une description détaillée dans la section 5.2.

5.1.3 Organisation mémoire

D'une manière générale, dans tout système informatique, la mémoire est représentée par un espace de données indexé par une adresse. Chaque adresse identifie un octet unique (8 bits) et le nombre de bits n que l'on attribue à l'adresse définit la taille de l'espace mémoire adressable (2^n octets). Il est à noter que l'espace d'adressage peut être plus grand que la taille de mémoire physiquement présente dans le système, il est donc important de distinguer l'espace de mémoire adressable de l'espace mémoire physiquement disponible.

Les composants ne travaillent pas en général sur des octets directement, mais sur des *mots*, constitués de m octets. Typiquement, l'adresse est codée sur 32 bits (ce qui représente 4Go d'espace adressable), et un mot est constitué de 4 octets (32 bits). La mémoire présente au sein d'un SoC est souvent segmentée en différents blocs (chaque bloc contenant un morceau de l'espace d'adressage totale). Ces blocs peuvent correspondre à des composants de mémorisation (appelés *mémoires*) ou à des registres de composants dont la fonction principale n'est pas la mémorisation (accélérateurs matériels, processeurs, périphérique). C'est l'interconnexion qui aura la charge de

transmettre les requêtes des différents composants à une adresse mémoire donnée au bon bloc de mémorisation (mémoire ou registre de contrôle).

Les périphériques et les accélérateurs matériels, qui n'exécutent pas de code directement, sont dit "assignés en mémoire" (*memory mapped* en anglais), c'est à dire qu'on les contrôle en écrivant dans des registres qui disposent d'une adresse dans l'espace d'adressage mémoire. Le *pilote* (*driver* en anglais) de ces composants est un morceau de code qui s'exécute sur un processeur de la puce, et dont la fonction est de lire et écrire les registres du composant pour lui faire exécuter la fonction désirée.

Sur la puce, on ne peut pas intégrer autant de mémoire que l' on veut car cela occupe beaucoup de surface de silicium. En général, toute la mémoire physique ne sera donc pas intégrée dans la puce, une partie sera stockée en dehors, dans des barrettes de RAM par exemple. Des interfaces d'accès à cette mémoire "externe" seront alors intégrées sur la puce. C'est par exemple le cas dans le SoC de la figure 5.2, qui contient deux interfaces d'accès à de la mémoire sur la carte. Bien entendu, il sera beaucoup plus rapide d'accéder à des données stockées dans les blocs mémoire sur la puce, que d'accéder à des données stockées à l'extérieur. La totalité des SoC à l'heure actuelle fonctionne avec un seul espace d'adressage. Les communications dans un SoC sont exclusivement constituées d'accès à la mémoire [30, 93].

5.1.4 Conception des SoC

Cette section présente rapidement la problématique et les enjeux de la conception de systèmes sur puce. Les critères de conception sont discutés dans la section 5.1.4.1, le flot de conception est décrit dans la section 5.1.4.2 et la réutilisation de composants est présentée dans la section 5.1.4.3. Enfin les techniques de simulation sont exposées dans la section 5.1.4.4.

5.1.4.1 Critères de conception des SoC

Un SoC est classiquement évalué selon quatre métriques :

- **Surface**. C'est la surface que va occuper le circuit sur le silicium (en mm^2). Elle est directement liée au nombre et à la complexité des composants intégrés sur la puce. Elle dépend aussi de la technologie (finesse de gravure) qui est, selon la loi de Moore, divisée par deux tous les dix-huit mois.
- **Performance**. C'est la puissance de calcul ou de traitement du système. On l'exprime en général par le nombre de cycles (ou le temps en intégrant la fréquence de l'horloge) pris pour faire un traitement donné.
- **Consommation**. C'est la consommation électrique du circuit. L'importance de la consommation dans la conception de systèmes sur puce a beaucoup grandi ces dernières années (afin de pouvoir concevoir des puces peu consommatrices qui peuvent être embarquées pendant une longue durée sans nécessité de recharge), c'est maintenant un critère déterminant (en particulier car l'évolution des technologie de batterie est bien plus lente que celle de l'intégration des transistors dans une puce). On distingue deux types de consommations : la consommation dynamique, qui est liée à l'activité du circuit (basculement des portes logiques) et la consommation statique qui correspond à des courants de fuite (même si le circuit ne fait rien, il consomme un peu d'énergie). Il est aussi important de noter la différence entre l'énergie moyenne consommée par unité de temps qui va directement déterminer la durée de vie d'un système sur batterie, et la puissance maximale qui

va engendrer un échauffement du circuit pouvant entraîner des dysfonctionnements.

- **Coût financier**. C'est un aspect incontournable. Il est fonction de la technologie, de la surface de silicium et du temps de conception.

Ces critères ne sont pas indépendants, ils sont même parfois antinomiques, et aucune puce ne pourra être optimale sur chacun d'entre eux. C'est pourquoi la conception de SoC est avant tout une affaire de compromis, à tous les niveaux de conception. Plus ces compromis sont situés à des niveaux d'abstraction élevés, plus les gains sont grands. Il est donc nécessaire d'élever le niveau d'abstraction dans le flot de conception. Découpler le calcul des communications permet par exemple non seulement de pouvoir interconnecter facilement des composants sur différents SoC, mais aussi de prendre des décisions plus tôt dans le flot de conception (alors qu'on ne sait pas encore quelle sera l'interconnexion). C'est pourquoi ce découplage favorise la réutilisation et permet aux concepteurs de faire des optimisations à de plus hauts niveaux d'abstraction, et donc de réduire le temps (et le coût) de conception.

5.1.4.2 Flot de conception

La conception de systèmes sur puce est un problème très complexe et un domaine très actif de recherche. Il faut dissocier deux grandes catégories afin de décrire le flot de conception en détails :

- **Le matériel**. C'est l'ensemble des composants intégrés dans la puce (processeurs, mémoires, accélérateurs, etc.).
- **Le logiciel**. Cela recouvre le code qui sera exécuté sur les processeurs, le système d'exploitation (OS pour *Operating System*), les codes pour piloter certains composants (*drivers*). Le logiciel peut être simplement un code séquentiel, ou bien être un ensemble de *processus* appelé *thread* en anglais. Nous utiliserons dans la suite de ce texte le terme "thread" pour s'y référer afin d'éviter la confusion avec les processus stochastiques.

Concevoir un SoC, c'est l'action de déterminer une architecture matérielle (ensemble des composants) ainsi qu'une organisation logicielle (systèmes d'exploitations, applications, etc.) permettant de remplir les critères fixés dans le cahier des charges du produit en cours de développement.

Sachant que certaines fonctions peuvent, soit être effectuées par du logiciel s'exécutant sur un processeur (lent mais flexible), soit exécutées par un composant dédié (rapide mais non programmable), une des problématiques majeures dans la conception de SoC est de faire ce que l'on appelle le partitionnement logiciel/matériel, c'est à dire de choisir les fonctions qui seront exécutées par des accélérateurs matériels, et les fonctions qui seront exécutées par du logiciel. On peut en général modéliser une application par un graphe de tâches communiquantes, le partitionnement logiciel/matériel consiste à affecter ces tâches sur les composants matériels de la future puce.

On parle donc souvent de *flot de conception en Y* (voir figure 5.4), car à partir de spécifications matérielles et logicielles (les deux branches supérieures), on recherche une solution en effectuant le partitionnement logiciel/matériel (regroupement des deux spécifications). On procède ensuite à une co-simulation du logiciel et du matériel afin de déterminer si les contraintes du cahier des charges en terme de surface de silicium, de consommation et de performances sont atteintes. Si ce n'est pas le cas on recommence le partitionnement.

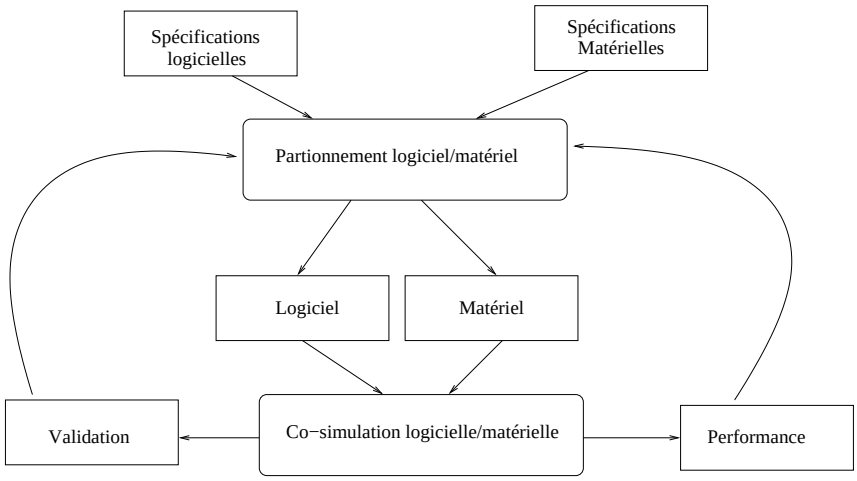


Fig. 5.4: Flot de conception de systèmes sur puce en Y.

On peut voir la conception de SoC comme la sélection d'un point dans l'espace de conception (*Design space* en anglais). Un point de cet espace correspondant à un système particulier (composants paramétrés, logiciel) que l'on peut évaluer par simulation sur les critères introduits précédemment (performance, surface, consommation). La taille de l'espace étant très grande et l'évaluation de chaque point prenant un temps important (voir section 5.1.4.4), il est impossible d'effectuer une recherche exhaustive du meilleur point, c'est à dire celui respectant le cahier des charges et minimisant le coût sur chacun des critères introduits à la section précédente. De nombreux projet de recherche visent à proposer des méthodes pour améliorer et automatiser la conception de SoC, le lecteur intéressé est renvoyé au livre [30] pour obtenir des détails et des références.

La conception de SoC fonctionne aussi par raffinements successifs, c'est à dire que l'on va, au fur et à mesure que le projet avance, effectuer des simulations de plus en plus précises (voir section 5.1.4.4) et prendre des décisions concernant des détails de plus en plus fins. Ceci implique que le flot de conception en Y n'est pas parcouru une unique fois, mais de nombreuses fois au contraire. A chaque étape des décisions de niveau de plus en plus bas sont prises, le SoC étant petit à petit raffiné pour converger vers le produit final.

Il est enfin à noter que la conception de tels systèmes est une activité qui occupe plusieurs centaines d'ingénieurs pendant plusieurs année. En particulier, le développement logiciel sur de telles plateformes est extrêmement important. Par exemple, la société Siemens employait 30000 personnes pour la conception de ses puces (en 2000), c'est à dire plus que Microsoft pour le développement de ses applications.

5.1.4.3 Réutilisation

Comme cela a déjà été mentionné plus haut, un SoC contient plusieurs millions de portes logiques interconnectées, et il est évidemment impensable de concevoir ces systèmes en décrivant directement cet assemblage de portes logiques. Il faut alors élever progressivement le niveau d'abstraction afin de masquer la complexité et de permettre à des hommes et des femmes de faire une conception efficace. Ces différents niveaux d'abstraction sont représentés sur la figure 5.5 pour différentes descriptions d'un système embarqué donné. Les niveaux d'abstraction des communications sont détaillés dans la section 5.1.4.4.

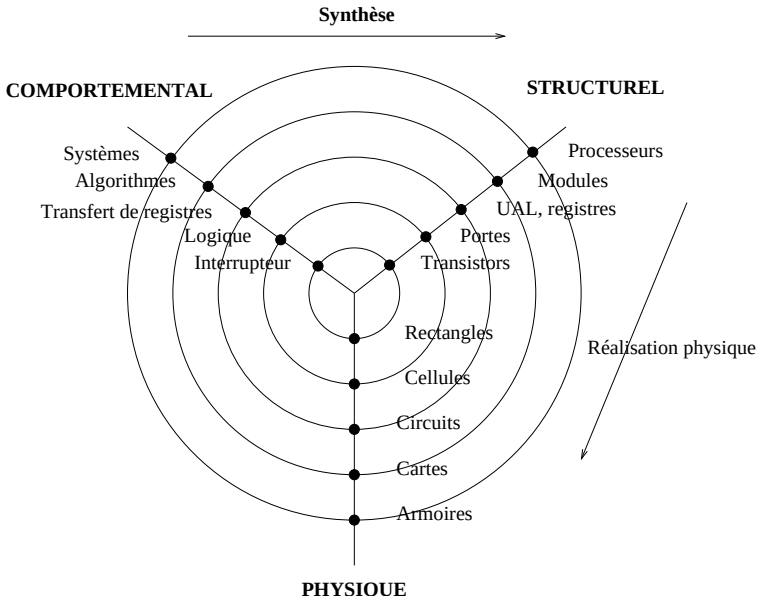


Fig. 5.5: Diagramme de Gajski représentant les différents niveaux d'abstraction pour différentes descriptions d'un système (comportemental, structurel et physique).

L'évolution des technologies suit, depuis quarante ans déjà, la *loi de Moore* (Le nombre de transistors par unité de surface double tous les 18 mois), qui est maintenue très rapide par les fondeurs, alors que les temps de conception suivent une évolution moins rapide. L'écart entre les technologies disponibles et leur utilisation augmente. Il est donc nécessaire d'aller vers des méthodologies visant à diminuer les temps de conception.

La complexité croissante des SoC pose aussi de grands problèmes de conception. En effet, alors que traditionnellement chacun d'entre eux était conçu indépendamment en utilisant une expertise acquise par les concepteurs, leur complexité est maintenant telle que cette méthode trouve ses limites. Les temps de simulation pour validation à chaque étape sont trop longs et les contraintes de temps avant mise en vente (*Time To Market*) sont trop fortes. Il est donc nécessaire de mettre en place de nouvelles méthodologies de conception qui doivent faire intervenir la réutilisation car c'est le seul moyen d'aller plus vite et d'élever le niveau d'abstraction pour réduire les temps de validation et de test. C'est pourquoi les concepts de composant virtuel (VC pour *Virtual Component*), qui correspond à une représentation réutilisable d'un composant a été introduit. Cette représentation est en général synthétisable. On utilise plus l'acronyme IP (pour *Individual Property*) pour parler de ces composants virtuels, qui en pratique peuvent être développés et vendus par des sociétés, ce qui introduit la problématique de la propriété intellectuelle. Par abus de langage, les composants virtuels seront appelés IP dans la suite de ce texte, car c'est la dénomination la plus souvent rencontrée dans la littérature. Les composants virtuels doivent bien entendu être paramétrables et facilement testables, mais il faut surtout pouvoir les interconnecter, ce qui justifie le rôle critique de l'interconnexion.

Pour que les composants puissent être réutilisés et agencés librement, ainsi que pour les rendre plus flexibles, il est impératif de découpler les calculs des communications. Ce découplage permet de pouvoir séparer la conception des IP et de l'interconnexion ainsi que de réutiliser une IP dans différentes puces. Pour cela, il a été nécessaire de définir des interfaces de communication standards. Deux initiatives ont été menées dans ce sens : VCI (*Virtual Component Interface*) [5] et OCP (*Open Core Protocol*) [113]. L'interface VCI est en train de disparaître au profit de OCP, néanmoins, comme dans nos expérimentations l'interface VCI est utilisée, la section 5.3.2

la décrira en détails. Il existe aussi beaucoup d'interfaces propriétaires (ST-Bus, Amba, etc.).

Nous allons donc vers des SoC *plug and play* où il suffira de choisir les composants (Processeur, Mémoire, DSP, interconnexion) dans une bibliothèque et de les assembler pour pouvoir réaliser un SoC [30]. La réutilisation de composants génériques (IP) est donc de manière évidente l'avenir des SoC.

Afin toujours d'essayer d'améliorer la réutilisation, on peut aussi chercher à restreindre l'espace de conception en travaillant sur des *plateformes* paramétrables [30]. Le concepteur choisira une plateforme adaptée à son besoin parmi un ensemble de plateformes et cherchera à la paramétrer pour que son application s'exécute en respectant le cahier des charges. Des plateformes pourront être développées pour chaque secteur de marché (téléphonie, multimédia, PDA, etc.). Ce nouveau paradigme de conception a émerger différents types de SoC, comme les SoC multiprocesseur par exemple. Ces derniers, regroupés sous l'acronyme MPSoC (pour *Multiprocessor SoC*) sont des architectures constituées d'un ensemble processeurs généralistes et de bancs mémoires, ainsi que d'entrées/sorties. Ce sont donc de véritables machines parallèles sur puce. Les cibles potentielles de ce type de puces sont par exemple les processeurs réseaux (processeurs des équipements de cœurs de réseaux), où la même application (routage, gestion de la qualité de service) est répliquée pour chaque port, et donc peut être affectée sur un processeur différent pour permettre le routage de plusieurs liens Gigabits en parallèle. Les MPSoC sont par nature très flexibles car l'ensemble des fonctionnalités est implémenté en logiciel. C'est pourquoi ils sont bien adaptées aux terminaux incluant des applications multiples et évolutives (agenda, jeux, audio, vidéo, téléphonie mobile de troisième génération, etc.).

5.1.4.4 Simulation et niveaux d'abstraction

La simulation de SoC peut être effectuée à différents niveaux de précision. Plus la simulation est précise, plus elle est longue et donc moins elle permet une exploration large de l'espace de conception. Le tableau 5.1 fait un résumé des différents types de simulation qui sont décrits ci-après :

- **Niveau processus communiquants**. C'est le niveau d'abstraction le plus élevé. On ne dispose ici d'aucune information sur les composants matériels qui seront utilisés, on décrit l'application comme un ensemble de tâches (fonctions) communiquant par des canaux abstraits point à point. On valide à cette étape le fonctionnement globale du système, mais on ne peut pas faire d'analyse de performance. Les simulations à ce niveau sont très rapides (de l'ordre de la vitesse réelle). Les processus de Kahn [74] est un exemple de modélisation de ce niveau.
- **Niveau transaction** (TLM pour *Transaction Level Modelling*). A ce niveau, la plateforme matérielle fait son apparition dans la simulation. On reste néanmoins à un niveau comportemental (le comportement des composants est correct, mais aucune information de temps n'est donnée, il n'y a pas d'horloge). Les composants communiquent par des canaux de communication abstraits. Cela permet de simuler rapidement pour développer la partie logicielle et les pilotes des périphériques par exemple.
- **Niveau transaction temporisée** (TLM-T pour *Timed* TLM). On ajoute à ce niveau des informations sur le temps que prennent les calculs et les communications sans que ceux-ci soient précisément simulés. On utilise par exemple, pour simuler un processeur, un ISS (*Instruction Set Simulator*) qui simule l'architecture interne du processeur (registres, accès mémoires, etc.), mais qui n'est pas précis au cycle près.

- **Niveau cycle** (CABA pour *Cycle accurate Bit accurate*). Ce niveau correspond à un modèle du système ayant le même comportement au cycle près localement (pour chaque composant), respectant les valeurs des *bits* d'informations qui circulent dans ce système. On peut pour cela utiliser un langage de description de matériel (VHDL, VERILOG) ou utiliser SystemC.
- **Niveau transfert de registres** (RTL pour *Register Transfer Level*). On décrit ici le système à l'aide de blocs de base (registres, additionneurs, etc.), dans un langage de description de matérielle comme VHDL ou VERILOG (sous-ensemble synthétisable). Cette description est *synthétisable*, c'est à dire qu'on peut, à partir de celle-ci, générer un circuit qui pourra être fabriqué.
- **Niveau portes logiques.** À ce niveau d'abstraction, le système est décrit comme une interconnexion de portes logiques. Étant donnée la complexité des SoC de nos jours, il est de plus en plus difficile de manipuler cette description du système. On préfère depuis quelques temps déjà la dériver automatiquement (cette étape fait partie de la *synthèse logique*) d'une description de niveau transfert de registres.
- **Niveau transistor.** On simule ici de manière physique le comportement de chaque transistor pour évaluer le temps de passage des portes et en déduire la fréquence à laquelle la puce va pouvoir être cadencée. Ces simulations peuvent être réalisées à l'aide d'outils tel que Spice [140]. Elles sont très lentes.

Niveau d'abstraction	Langage associé	Logiciel	Vitesse (cycles/seconde)
Processus communiquants	C/C++	-	$\sim 10^7$
TLM	C/C++/SystemC	SystemC	$\sim 10^6$
TLM-T	C/C++/SystemC	SystemC	$\sim 5 \times 10^5$
CABA	VHDL/SystemC	SystemC	$\sim 10^4$
RTL	VHDL/SystemC	ModelSim	$\sim 10^3$
Porte logique	VHDL/VERILOG	ModelSim	~ 10
Transistor	netlist	Spice	~ 1

Tab. 5.1: Simulation des différentes représentations d'un SoC à différents niveaux d'abstraction. Les langages et logiciels sont donnés à titre indicatif, de nombreux autres langages/logiciel existent.

On peut résumer l'effort pour combler l'écart entre l'évolution des technologies et l'évolution de la rapidité de conception comme la définition de nouveaux niveaux d'abstraction. En effet, environ tous les dix ans, un niveau d'abstraction supérieur est défini à partir duquel une grande partie des choix de conception peut être faite. On peut ainsi effectuer des simulations précises de plus haut niveau et donc arriver à concevoir les SoC plus rapidement. La définition du niveau RTL date des années 1990 et la définition du niveau TLM (dernier niveau d'abstraction à partir duquel on peut raffiner jusqu'à la synthèse) date de 2000. La synthèse et la conception à partir du niveau processus communiquant est aussi un domaine actif de recherche appelé *synthèse de haut-niveau*, néanmoins ce problème est encore loin d'être résolu.

5.2 Réseaux sur puce

Cette section présente un état de l'art de différentes propositions de réseaux d'interconnexion pour SoC. L'interconnexion des composants intégrés dans une puce est, à l'heure ac-

tuelle, en train de connaître un changement technologique majeur avec l'apparition de réseaux sur puce en remplacement des traditionnels bus partagés. Ces bus font l'objet de la section 5.2.1, puis la section 5.2.2 explique les raisons de l'émergence de véritables réseaux sur puce et la section 5.2.3 décrit les propositions les plus avancées.

5.2.1 Bus et bus hiérarchiques

Pour faire communiquer entre eux les différents éléments d'une puce (mémoires, processeurs, accélérateurs, etc.), c'est le modèle à bus partagé qui a, dans un premier temps, été utilisé. Les bus sont basés sur la diffusion de l'information sur un support partagé par tous les éléments communicants comme cela est illustré sur la figure 5.6. L'accès à ce support est géré par un arbitre, qui s'assure que tous les éléments puissent communiquer, en fonction éventuellement de leur priorité.

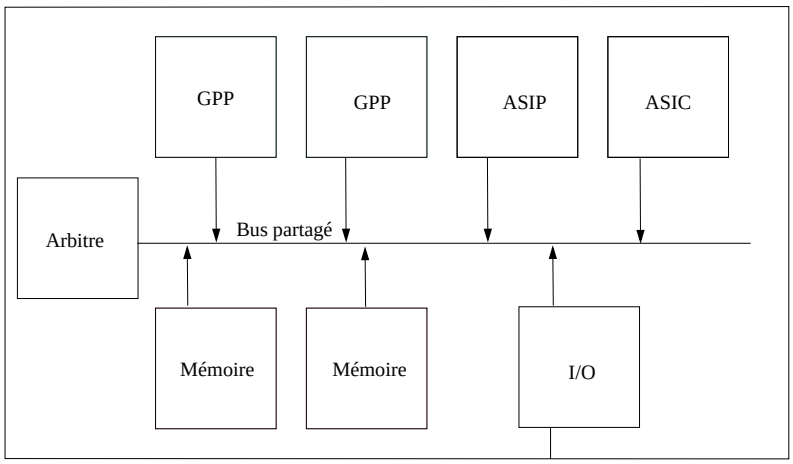


Fig. 5.6: Schéma d'un système sur puce avec un bus partagé classique.

Des fils séparés sont utilisés pour faire transiter l'adresse, les données et le contrôle. Cela permet aux bus de fonctionner en *pipeline* et donc d'améliorer leurs performances. Mettre en place un *pipeline* consiste à identifier des fonctions élémentaires (envoi de l'adresse, autorisation d'accès au bus, envoi et réception des données, etc.) pouvant être effectuées en même temps grâce aux différents fils. À chaque instant certaines fonctions seront effectuées simultanément. On tâche alors d'exploiter au maximum ce parallélisme pour masquer la latence de l'interconnexion et améliorer les performances. Les performances sont donc multipliées par le nombre de fonctions élémentaires appelées *étages* du pipeline. Pour augmenter le nombre d'étages, on peut par exemple augmenter le nombre de fils et en utiliser des différents pour la gestion des lectures et des écritures. D'une manière générale les requêtes et les réponses sont gérées ensemble (le bus est attribué par l'arbitre pour un couple requête/réponse) , mais certains bus implémentent aussi les transactions séparées ou *Split Transactions*, c'est à dire que les requêtes et les réponses sont gérées de manière indépendante. Ainsi si un composant met du temps à répondre, l'arbitre du bus peut accorder le bus à une autre communication. Le bus est une interconnexion synchrone, c'est à dire que tous les composants doivent être synchronisés entre eux et surtout avec l'arbitre. Il nécessite donc une horloge commune. Les interconnexions à base de bus sont simples à mettre en place, il suffit que tous les éléments possèdent une interface pour accéder au bus. On distingue deux types d'interfaces :

- **Initiateurs**. Ils sont les seuls à avoir accès au bus d'adresse en tant qu'initiateur. Ils peuvent uniquement faire des requêtes vers des récepteurs. Ce sont typiquement les processeurs, les composants dédiés disposant d'un DMA.
- **Récepteurs**. Ils ne peuvent que répondre aux requêtes d'un initiateur et piloter le bus de données. Ce sont typiquement les bancs mémoires et les périphériques.

Un exemple typique de bus classique est le PI-BUS (*Peripheral Interconnect Bus* [115]) qui a été déployé dans de nombreuses puces depuis 1995. Il est composé d'un bus de données de 32 ou 64 bits, et d'un bus d'adresse de 32 bits. L'arbitrage peut être adapté au système et inclure par exemple une gestion de priorités.

Néanmoins, un bus classique possède de nombreuses limites dues à sa simplicité :

- **Le nombre de composants raccordés.** L'arbitrage centralisé constitue un goulot d'étranglement. De plus la bande passante n'est pas fonction du nombre d'éléments. Le bus ne passe pas à l'échelle, c'est à dire que son architecture n'évolue pas en fonction du nombre de composants raccordés.
- **La distribution d'horloge.** La distribution de la même horloge à tous les éléments de la puce devient problématique avec les fréquences d'horloge actuelles.
- **La longueur des fils.** Les fils longs posent trois problèmes principaux détaillés ci-après :
  - Pour augmenter les performances et diminuer la consommation des SOC, la tension d'alimentation de ceux-ci baisse (elle est passée de 4V à moins de 1V en 5 ans). Ceci augmente les erreurs de transmission, et bride donc les performances du bus. Dans un réseau, les fils sont moins longs car l'arbitrage est distribué.
  - La consommation des fils est proportionnelle à leur longueur (car l'impédance augmente avec la longueur), et malgré toutes les techniques de codage utilisées pour réduire l'activité électrique [134, 95], le bus reste un grand consommateur d'énergie. Un réseau étant constitué de petites liaisons, il consomme donc naturellement moins qu'un bus.
  - L'augmentation des fréquences d'horloge rend la synchronisation de plus en plus difficile. Tout le monde s'accorde à concevoir des systèmes localement synchrones et globalement asynchrones. Le temps de propagation du signal dans les fils devient proche du temps d'horloge, et la distribution de l'horloge sur la puce devient problématique dans le cas synchrone.

Afin de pouvoir répondre en partie à ces limites, des bus hiérarchiques ont été développés. Il s'agit de mettre plusieurs bus reliés entre eux par des ponts. Ainsi le nombre d'éléments par bus est réduit, la consommation aussi car les fils sont plus courts, et la distribution de l'horloge est moins problématique. Chaque bus possède son propre arbitre et les bus sont reliés entre eux par des ponts qui appartiennent à une paire de bus. A partir d'une estimation des flux, le concepteur pourra à sa guise regrouper les éléments qui demandent un haut débit sur des bus hautes performances, ceux qui ont un débit plus faible sur des bus périphériques, et établir les connexions entre les différents bus. On se rapproche donc d'un réseau où les ponts sont simplement des routeurs extrêmement simplifiés. Le SOC de la figure 5.2 dispose par exemple d'un bus système et d'un bus périphérique. La spécification AMBA [32] définit par exemple trois bus :

- **Un bus très hautes performances** (AHB, *Advanced High-Performance Bus*) possédant un bus de données large (128 bits) et pouvant fonctionner à des fréquences élevées.
- **Un bus moyennes performances** (ASB, *Advanced System Bus*) possédant des caractéristiques un peu moins avancées que le AHB.
- **Un bus périphérique** (APB, *Advanced Peripheral Bus*) pour connecter les composants bas

débits et faible consommation.

Le tableau 5.2 présente les caractéristiques principales de cinq bus utilisés actuellement sur puce. Tous ces bus ont des bus d'adresses de 32 bits. Les bus hiérarchiques seront classés du plus performant au moins performant. Les bus SuperH SuperHighway [143] et IBM CoreConnect [64] sont semblables aux bus AMBA. Nous avons calculé le débit utile pour un taux d'utilisation de 40%, ce qui est représentatif d'une interconnexion bien dimensionnée [57]. Une utilisation supérieure engendre en effet une latence trop importante.

Bus	Fréquence	Bus de données	Débit utile	Arbitrage	Interface
Pi-Bus	100 MHz	64 bits	2.56 Gb/s	centralisé	Propriétaire
AMBA HB	200 MHz	128 bits	10.2 Gb/s	centralisé	OCP
AMBA SB	100 MHz	64 bits	2.5 Gb/s		
AMBA PB	50 Mhz	32 bits	0.6 Gb/s		
IBM CoreConnect 1	183 MHz	128 bits	9.4 Gb/s	centralisé	OCP
IBM CoreConnect 2	133 MHz	64 bits	3.4 Gb/s		
IBM CoreConnect 3	66 MHz	32 bits	0.8 Gb/s		
SuperHighway 1	200 MHz	1024 bits	82 Gb/s	centralisé	VCI
SuperHighway 2	100 MHz	256 bits	10.2 Gb/s		
SuperHighway 3	50 MHz	64 bits	1.2 Gb/s		
Silicon backplane	NC	64 bits	NC	TDMA	OCP

Tab. 5.2: Tableau comparatif des différents bus existants.

La colonne interface correspond à l'interface standard implémentée de manière native par les bus, c'est à dire pour laquelle il n'y a pas besoin d'adaptation. Le débit du *SuperHighway 1* est très grand à cause de la largeur maximale de son bus de données (1024 bits). Une telle utilisation est réservée à des cas spécifiques, dans une utilisation normale la taille des paquets est largement inférieure à 1024 bits. Le constructeur conseille d'ailleurs une utilisation à 256 bits, ce qui ramène le débit à 20,4 Gbs.

5.2.2 Vers les réseaux sur puce

L'avantage du bus est principalement sa simplicité, et donc son faible coût. Le principe de diffusion permet aussi de régler facilement les problèmes de cohérence de mémoires caches. Cependant les limitations des bus décrites dans la section précédente, ont été mises en évidence par la demande en débit et latence des applications flot de données et par l'augmentation du nombre de composants sur les puces. Il est donc nécessaire, pour les SoC à venir, de concevoir des interconnexions extensibles, très performantes, peu consommatrices mais aussi compatibles avec les modèles de communications existants. La mise en place de réseaux sur puce où plusieurs communications peuvent être gérées en parallèle semble donc une évolution logique. Les fils sont plus courts et consomment donc moins, la bande passante globale augmente, la localité est exploitée, et le passage à l'échelle vis-à-vis du nombre de composants intégrés est garantie.

Un réseau sur puce (NoC pour *Network on Chip*) est, comme un réseau de machine, constitué d'un ensemble d'éléments de routage (commutateurs ou *switch* en anglais) interconnectés par des liens (ensemble de fils) comme cela est illustré sur la figure 5.7. Il n'y a plus dans ce cas de fils séparés pour les adresses et les données, tout transite sur les mêmes liens. Les commutateurs sont de petits composants disposant de *ports* d'entrées sorties et leur rôle est simplement

de transmettre les données arrivant sur les différents ports d'entrées, vers les différents ports de sortie.

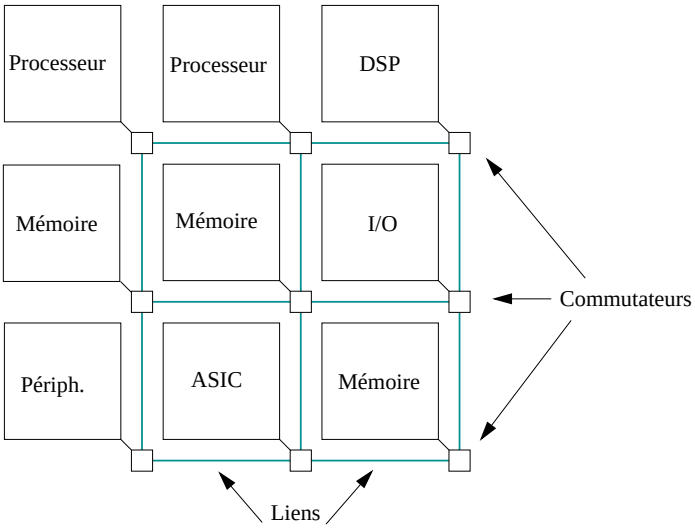


Fig. 5.7: Exemple d'un SoC avec un réseau sur puce ayant une topologie en grille 2D.

Les NoC peuvent être séparés en deux grandes catégories : les réseaux à *commutation de circuit*, assimilables aux réseaux téléphoniques dans lesquels chaque connexion se voit attribuer un chemin réservé de bout en bout, et les réseaux à *commutation de paquets*, comme le réseau Internet par exemple, ou les données sont envoyées sous forme de paquets de taille potentiellement variable qui peuvent prendre des chemins différents dans le réseau. Un paquet est découpé en unités de transfert indivisibles appelées *flit* pour *Flow Transfer Unit*, correspondant en général à la largeur (nombre de fils) des liens du NoC. Voici une liste des différentes caractéristiques d'un NoC :

- **Topologie**. C'est l'organisation des commutateurs (nombre, nombre de ports, liens, placement des ressources). Il existe de nombreuses topologies comme la grille qui est illustrée sur la figure 5.7, les arbres, les tores, etc.
- **Routage**. C'est l'algorithme qui va déterminer comment les paquets sont orientés dans le réseau. Chaque commutateur, lorsqu'il reçoit un paquet sur l'un de ces ports d'entrée, a pour unique rôle de le transmettre sur un port de sortie. L'algorithme qui associe à un paquet et un port d'entrée un port de sortie est appelé algorithme de routage et dépend en général de la topologie. Le routage peut être soit déterministe (un paquet empruntera toujours le même chemin pour aller d'un point à un autre), soit non-déterministe (l'état du réseau est pris en compte dans la décision de routage).
- **Mémorisation**. C'est la politique de mémorisation des paquets dans les routeurs. Il en existe de différents types : mémorisation en entrée (avant le routage), en sortie (après le routage). On peut aussi faire transiter les paquets en "trou de ver" (*wormhole* en anglais), c'est à dire que l'on pourra commencer à transmettre le paquet sur le port de sortie même si le dernier *flit* de celui-ci n'est pas encore arrivé.

Ces caractéristiques vont différencier les différents projets de réseaux sur puce. L'espace de conception des NoC étant beaucoup plus grand que celui des bus (même hiérarchiques), un réseau sur puce vient donc généralement avec une méthodologie de conception adaptée.

Malgré tous les avantages que l'arrivée des réseaux sur puce permet d'obtenir, elle introduit néanmoins de nouveaux problèmes :

- **Cohérence des caches**. De nouvelles méthodes de gestion de cohérence des caches devront être mises en place puisque les composants n'auront pas accès à l'intégralité du trafic (ce point sera discuté dans la section 5.3.3.2).
- **Ordre des communications**. Les réseaux à commutation de paquet peuvent ne pas conserver l'ordre temporel des communications. Dans ce cas, il faut mettre en place des mécanismes pour les réordonner car pour garantir la consistance mémoire, les requêtes de lecture et d'écriture doivent impérativement arriver dans l'ordre où elles ont été émises.

Historiquement, le premier réseau sur puce développé est PROPHID [87] (Philips Research) qui a été conçu pour une application gérant un affichage multi-fenêtres pour télévision numérique. Tous les composants sont connectés à un réseau composé de commutateurs (*crossbar*). Pour chaque connexion une liaison est établie de bout en bout pendant sa durée de communication (commutation de circuits). Ainsi, la bande passante et la latence sont garanties. Ce réseau a été conçu pour gérer des communications de type flot de données, c'est pourquoi dans l'architecture PROPHID, un bus de contrôle (qui n'a pas besoin d'être très performant) se charge du trafic de contrôle.

Néanmoins, il est apparu que PROPHID n'était pas assez flexible. En effet, d'une part chaque composant possède deux interfaces (bus de contrôle et PROPHID) et la logique nécessaire pour synchroniser l'ensemble occupe une grande surface. D'autre part, le fait que la bande passante soit réservée pour les communications permet d'avoir un modèle de communication déterministe mais peu flexible. La bande passante allouée est perdue si elle n'est pas utilisée, il faut donc que le réseau soit conçu avec une bonne connaissance des flux. Ces limites sont inhérentes à la commutation de circuit mise en place, et en se basant sur les limitations de PROPHID, un groupe de chercheurs de l'équipe ASIM du LIP6 a élaboré une architecture à commutation de paquets : SPIN[1] [57, 54, 103]. La quasi-intégralité des travaux en cours sur les NOC utilise maintenant la commutation de paquet.

5.2.3 Quelques réseaux sur puce

Cette section présente quelques propositions importantes de réseaux sur puce. Après avoir introduit les travaux fondateurs dans la section 5.2.3.1, trois propositions de réseaux qui correspondent aux plus avancées à notre connaissance seront détaillées : SPIN et DSPIN dans la section 5.2.3.2, NOSTRUM dans la section 5.2.3.3 et Ætheral dans la section 5.2.3.4. Les autres travaux en cours seront rapidement décrits dans la section 5.2.3.5.

5.2.3.1 Travaux fondateurs

En parallèle avec les travaux du LIP6 (réseau SPIN), des études générales sur l'émergence des réseaux sur puce ont été faites. Giovani De Micheli a exposé de manière claire les enjeux des NOC [18, 160]. Son groupe de recherche a effectué des études générales sur les NOC, avec une activité plus centrée sur la consommation. Ils ont en particulier mis au point un modèle de consommation des commutateurs [16, 20, 17], et l'ont utilisé pour étudier l'influence de la taille des paquets sur la consommation du réseau [160]. Les simulations ont été faites dans le cadre de puces *multiprocesseur* (MPSOC).

[1] Scalable Programmable Interconnexion Network

W. J. Dally a aussi été un pionnier des NoC, comme le montre l'article "*Route Packets, not Wires, On-chip Interconnexion Networks*" [35], qui introduit le concept de NoC ainsi que la topologie en grille qui est maintenant de loin la plus utilisée. Les études menées par son groupe de recherche sont très inspirées de l'optimisation des réseaux hautes performances et concerne notamment les algorithmes de routage, ainsi que les architectures passant à l'échelle. Par exemple dans [123], un système de gestion de flux a été proposé : le *Flit-Reservation Flow Control*. Ce système a été pensé pour les réseaux sur puce, mais il est clairement inspiré de leurs études sur les réseaux hautes performances. L'idée est de réserver des *buffers* avant que les données n'arrivent dans les commutateurs. Pour cela des *flits* de contrôle sont transmis avant les *flits* de données, sur un réseau différent. Ces *flits* de contrôle ont pour rôle d'annoncer l'arrivée d'un certain nombre paramétrable de *flits* de données, afin que le commutateur puisse leur réserver de la place. Les simulations montrent que ce système est plus performant que la stratégie "trou de ver" (*wormhole*) utilisée dans de nombreux NoC. En revanche, aucune information concernant la complexité (surface occupée par la logique dans les commutateurs et par le réseau parallèle) ni la consommation n'est fournie, la préoccupation de ces contributions étant uniquement la performance.

5.2.3.2 SPIN et DSPIN

Le réseau SPIN [57, 54, 103] dispose d'une topologie en arbre élargi (qui passe à l'échelle par nature) et utilise une mémorisation de type "trou de ver". Il est compatible, à un coût très faible, avec l'interface standard VCI. Au départ SPIN proposait deux politiques de routage : un routage dynamique (les tables de routage évoluent au cours du temps, mais les paquets peuvent se doubler) et un routage statique. Le routage dynamique imposait un réordonnancement des paquets à l'arrivée, c'est pourquoi il a été abandonné. En effet le gain en performance ne compensait pas le surplus de logique nécessaire à la mise en place de ce système. Aussi il a été décidé, afin d'éviter les inter-blocages, de faire passer les requêtes et les réponses dans deux sous-réseaux virtuellement disjoints.

Une nouvelle version de SPIN est actuellement en développement, il s'agit de DSPIN (*Distributed SPIN* [103]), qui est une grille 2D munie d'un routage déterministe simple (XY). Les liens bidirectionnels entre les routeurs sont asynchrones. Ce réseau a été pensé pour les grands SoC, et chaque élément raccordé au réseau peut être en fait un sous-système comportant plusieurs composants reliés entre eux à l'aide d'une interconnexion locale (un bus par exemple).

SPIN et DSPIN sont intégrés dans SocLib [114], ont peut donc les simuler au cycle près en SystemC et nous avons effectué nos expérimentations sur ces réseaux (voir chapitre 8).

5.2.3.3 Nostrum

Nostrum est un réseau sur puce développé en Suède (*KTH*, Stockholm), dans le cadre d'un projet en partenariat avec de nombreuses entreprises comme Tekes, Vinnova, Nokia et Ericsson. Ce projet a pour objectif le développement complet d'une architecture de NoC. Leurs travaux adressent de nombreux problèmes tels que la topologie [69], la pile protocolaire [107], la qualité de service [106] ainsi que l'évaluation de performance par la simulation [141, 142, 145, 146]. Nostrum dispose d'une topologie en grille 2D, où chaque IP possède un commutateur lui permettant de se connecter au réseau pour pouvoir communiquer avec les autres IP [107]. Comme dans le cas de DSPIN, chaque cellule peut être soit une IP, soit un ensemble d'IP reliés par un bus. La méthodologie de conception du réseau est contrée sur le placement des IP pour limiter

le nombre de communications inter-cellules, mais aussi pour réduire la taille des chemins entre les entités qui communiquent beaucoup. L'algorithme de routage est le *deflective routing* [46] qui permet d'utiliser très peu de mémoire dans les commutateurs. A chaque cycle, le nombre de *flits* entrants et de *flits* sortant est le même, c'est à dire que tous les *flits* entrant sont transmis vers une sortie, même si elle ne correspond pas au chemin optimal pour que le *flit* atteigne sa destination. Ce système est très avantageux sur puce ou la surface est une contrainte. Cela permet aussi de répartir la charge, à condition que le routage soit effectué a l'aide d'informations sur l'état de congestion de ses voisins.

Deux types de communication cohabitent dans le réseau : le trafic *best effort* (pas de qualité de service) et le trafic à bande passante réservée par circuits virtuels (basé sur un système de division de temps TDMA modifié). Le service de garanti de bande passante et de latence repose sur deux concepts [106] : Les réseaux temporellement disjoints (TDN pour *Temporally Disjoint Networks*) et les conteneurs de boucles.

- **Réseaux temporellement disjoints**. Les TDN proviennent de la topologie (grille 2D) et du *deflective routing*. En effet, soit deux paquets émis aux cycles t_1 et t_2 depuis les ressources de coordonnées (x_1, y_1) et (x_2, y_2), on peut montrer que pour certaines valeurs t_1, t_2, (x_1, y_1) et (x_2, y_2), les deux paquets ne se rencontreront jamais. Cela fait apparaître deux réseaux temporellement disjoints, qui pourront être utilisés pour mettre en place des circuits virtuels. On peut augmenter le nombre de TDN en introduisant de la mémorisation (*buffers*) dans les commutateurs. On peut alors montrer que le nombre de TDN est $2\times T$ (T est taille des *buffers*).
- **Conteneurs de boucles**. La deuxième approche pour réaliser des circuits virtuels est de créer un conteneur de paquet qui tourne sans arrêt entre la source et la destination, et qui est prioritaire par rapport au trafic *best effort*. Ainsi l'émetteur aura une sortie réservée à chaque fois qu'un container reviendra de la destination.

Le combinaison de ces deux techniques permet d'obtenir $2T$ circuits virtuels. La bande passante au sein de chaque circuit virtuel pourra quant à elle être répartie entre les différentes communications en utilisant les conteneurs de boucles. La granularité de cette répartition dépend du temps d'aller retour entre la source et la cible. Les circuits virtuels doivent être fixés à la conception (pour configurer les commutateurs). Il est possible d'augmenter la bande passante des circuits virtuels, mais sans aucune garantie. Ce système suppose donc une bonne connaissance des différentes communications entre les composants, afin de guider la configuration des circuits virtuels.

Un simulateur de ce réseau à été développé et testé sur une application fournie par un industriel (Ericsson). Les résultats semblent montrer l'utilité de la décomposition en couche, mais la couche transport (incluant les mécanismes de qualité de service décrits plus haut) n'est pas encore simulée à notre connaissance. Ce simulateur [141] était dans un premier temps une extension de NS-2, un simulateur de réseau de machine présenté dans un cadre différent dans la première partie de ce manuscrit (voir section 4.3.2). Cependant cette extension restait très simpliste (UDP comme protocole réseau, paquets de taille fixe, etc.) et logiquement assez éloignée des contraintes spécifiques aux réseaux sur puce. Un autre simulateur a donc été développé [145, 146], utilisant une simulation basée sur le temps (*clock Synchronous Model*), dans lequel tout calcul prend 1 cycle et tout transfert sur un lien est instantané (proche du niveau transaction temporisées (voir section 5.1.4.3). L'ensemble devrait maintenant être migré en SystemC. Dans [146] Nostrum a été comparé avec un modèle de bus embarqué. Les résultats montrent que même si la latence est plus élevée avec Nostrum (les commutations doivent passer plusieurs commutateurs), il permet d'effectuer le traitement 9 fois plus vite qu'avec un bus,

car plusieurs commutations peuvent avoir lieu en parallèle.

Des simulations de NOSTRUM ont été récemment effectuées avec des charges aléatoires [147]. L'arrivée des paquets dans le réseau suit une loi de Poisson (voir section 1.3.3.2). Ces simulations ont montré que les circuits virtuels n'affectaient pas beaucoup le trafic *best effort*, et ce pour un faible sur-coût en surface.

5.2.3.4 Æthereal

Philips Research est historiquement le premier à avoir conçu un réseau sur puce à commutation de circuits appelé PROPHID [87]. En pratique il était trop peu flexible pour être pleinement adopté. C'est pourquoi Philips étudie maintenant un réseau à commutation de paquet (Æthereal [131, 133]). Ce réseau a dès le départ affiché la volonté de mettre en place de la qualité de service (QOS) afin de pouvoir par exemple garantir une bande passante pour certaines communications prioritaires [133]. Æthereal est développé dans le cadre d'applications multimédia ou de nombreux traitements intensifs et temps réel sont effectués.

La topologie de Æthereal est une grille 2D, avec une politique de routage par unité de temps appelé *slots*. Un *slot* est une période de quelques cycles allouée pour une communication entre deux composants. Une table d'attribution centralisée de ces *slots* aux différentes communications est configurée statiquement (lors de la conception du réseau), et on peut ainsi garantir pour des communications une bande passante en leur allouant un certain nombre de *slots*. Les *slots* non attribués sont utilisés pour faire passer les autres communications au mieux (*best effort*, pas de priorité sur les communications). Il est envisagé que la table des *slots* puisse être configurée pendant l'exécution, on peut alors considérer que Æetheral est un réseau à commutation de circuit reconfigurable, cependant il semble que ce ne soit pas encore implémenté.

5.2.3.5 Autres travaux

De très nombreuses propositions et études sur les réseaux sur puce ont vu le jour ces cinq dernières années (par exemple SOCBUS [92, 153, 154] à l'université de Linkoping, LOTTERYBUS [81] à Princeton , PROTEO [139] à l'université de Tempere et BONE [85, 86] en Corée du Sud). Certains travaux visent aussi à développer des méthodes pour générer automatiquement un réseau sur puce à partir d'une spécification de l'application [4, 128, 127]. Le compilateur de réseaux sur puce XPIPES [68, 21] est une initiative similaire.

Du coté des industriels, des travaux sont en cours à l'IMEC, qui développe un NOC pour interconnecter des blocs reconfigurables dynamiquement [100, 101]. ST Microelectronics possède aussi une activité importante [76, 102, 33] mais leurs travaux restent assez protégés. Les sociétés Sonics [65] et Arteris [135] commercialisent des solutions à base de réseaux sur puce.

On peut enfin citer deux initiatives relativement en marge de l'activité autour des NOC : Les réseaux sur puces asynchrones (comme CHAIN [10, 9] par exemple) et les interconnexions optiques [24, 23].

Deux sites Internet proposent des bibliographies très complètes des travaux réalisés dans le domaine des NOC :

- http://www.cl.cam.ac.uk/users/rdm34/onChipNetBib/browser.htm
- http://www.ocpip.org/university/biblio_main/

5.2.3.6 Tableaux récapitulatifs

Nom	Commutation	Topologie	Routage	QoS	État
SPIN (Paris)	Paquets	Arbre élargi	Statique	-	Synthèse[a]
CHAIN (Manchester)	Paquets (Asynchrone)	Hiérarchique	Mux/Demux	en cours	Fabrication[b]
Nostrum (Stockholm)	Paquets	Grille 2D	Deflectif	Circuits virtuels	simu HDL
IMEC (Leuven)	Paquets	Tore 2D	Wormhole	Circuits virtuels	Synthèse FPGA
Æthereal (Amsterdam)	Paquets	grille 2D	Slot	Circuits virtuels	?
BONE (Corée)	Paquets	2 crossbars	commutateurs	-	Fabrication[c]
SOCBUS (Linkoping)	PCC[d]	grille 2D	Statique	Connexions	Fabrication[e]

Tab. 5.3: Quelques propositions de réseaux sur puce.

[a] Fabrication partiel du réseau
[b] Fabrication des commutateurs
[c] Fabrication d'une puce simple (1 initiateur)
[d] Packet Connected Circuits
[e] Fabrication de commutateurs simplifiés (1 initiateur)

Voici des tableaux récapitulatifs résumant les différentes solutions de réseaux sur puce existantes. Le tableau 5.3 rassemble les principales caractéristiques (routage, mémorisation, topologie, qualité de service et état d'avancement) des différentes propositions de réseaux sur puce.

Nom	Réseau	Type	Code	Niveau	Trafic
SocLib	SPIN, DSPIN	Système sur puce	SystemC	CABA	Réel
SimLab	Paderborn	Réseaux de stockage	C++	TLM	Aléatoire
-	Chain	Réseaux asynchrones	JAVA	TLM	Aléatoire
-	NosTrum	réseaux sur puce	SystemC	TLM	Aléatoire
Rsim	Standford	Multiprocesseur	?	?	Réel
NoCSim	Texas	réseaux sur puce	SystemC	TLM-T	Traces
-	Linkoping	NoC	C++	CABA	Aléatoire
Ptolemy	Berkeley	Simulation de circuit	C++/Java	TLM	Tous

Tab. 5.4: Quelques simulateurs de réseaux sur puce.

Le tableau 5.4 établi quant à lui un descriptif des simulateurs qui ont été développés par la communauté scientifique pour évaluer les réseaux sur puces. En général, une équipe proposant un NoC développe aussi les modèles de simulations et un environnement complet de simulation dédiée pour en évaluer les performances.

Enfin, le tableau 5.5 compare les informations de surface disponible pour chacun de ces réseau. La surface équivalente a été calculée en utilisant le rapport des densités de portes entre les différents procédés (source ST Microelectronics). Ces chiffres sont à manipuler avec précaution car l'erreur engendrée par ces approximations peut être potentiellement très grande.

5.3 Trafic sur puce

Cette section traite des communications qui ont lieu sur puce, qui ont été le principal sujet d'étude de cette partie de la thèse. La section 5.3.1 présente des généralités, la section 5.3.2 décrit

Réseau	Année	Composant	Procédé	Surface	Surface
SPIN	2000	Routeur	0.18	0.4 mm^2	0.2 mm^2
	2000	Interface	0.18	0.4 mm^2	0.2 mm^2
	2000	Total	0.18	10-20 mm^2	5-10 mm^2
	2003	32 ports	0.13	5 mm^2	5 mm^2
CHAIN	2002	*Switch*	NA	350 T	437 nm^2
	2002	P2P + inter.	NA	30 kT	0.037 mm^2
IMEC	2002	Interface	NA	260 Slices	-
	2002	Router	NA	223 Slices	-
IMEC	2003	Interface	NA	661 Slice	-
Æthereal	2003	*Switch*	0.13	0.25 mm^2	0.25 mm^2
Linkoping	2003	*Switch* simple	0.8	0.5 mm^2	-

Tab. 5.5: Estimation de surface de différents réseaux sur puce. La surface est donnée en surface équivalente en technologie 130 nm.

en détails l'interface VCI et enfin la section 5.3.3 présente les différents types de trafics associé aux différents composants d'un SoC.

5.3.1 Généralités

Le trafic au sein d'un SoC est l'ensemble des communications qui ont lieu entre les différents composants. Nous feront l'hypothèse (justifiée car c'est le cas dans tous les SoC) que l'ensemble des données font partie d'un espace d'adressage unique.

Les communications sont des requêtes de différents types dont voici les deux principaux :

- **Lecture.** Un composant a besoin d'une donnée située à l'adresse D. Il effectue alors une requête de lecture à l'adresse D et le composant qui détient la donnée lui répond la valeur stockée dans cette case mémoire.
- **Écriture.** Un composant qui veut modifier la valeur d'une case mémoire située à l'adresse D. Il effectue alors une requête d'écriture à l'adresse D et le composant qui détient cette donnée modifie la valeur de la case mémoire correspondante. Ce dernier peut alors envoyer un acquittement (avec éventuellement un code d'erreur) au composant A, mais ce n'est pas obligatoire.

Des variantes de ces deux requêtes de base existent. On peut par exemple définir une requête de lecture/écriture atomique à l'adresse D, c'est à dire qu'aucune autre requête ne peut avoir lieu entre la lecture et l'écriture. Ce type de requête est utilisé pour la gestion des sémaphores par exemple. On peut aussi, afin d'accélérer les communications, faire des requêtes groupées, c'est à dire que l'on va effectuer une série de lectures ou d'écritures à n cases mémoire. Ce type de requête est appelée rafale (*burst* en anglais).

5.3.2 Interfaces standards

Pour effectuer leurs communications, chaque composant d'un SoC dispose d'une *interface*, c'est à dire un petit circuit qui gère le protocole de communication du système d'interconnexion (bus ou réseau). Comme cela a été introduit dans la section 5.1.4.3, des interfaces de communication standards ont été définies par des consortium d'industriels. Nous présentons ici l'interface VCI, qui est utilisée dans nos expérimentations.

Une interface de communication définit un protocole de communication point à point, afin

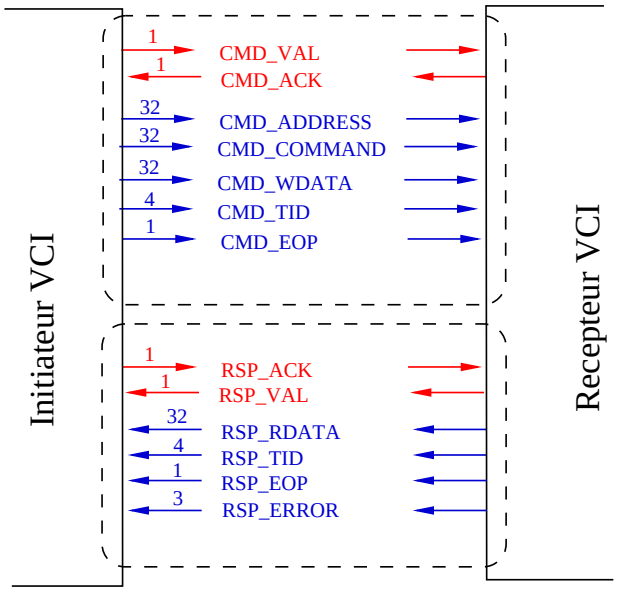


Fig. 5.8: Fonctionnement de l'interface standard VCI *Advanced*. Les nombres correspondents à la largeur (en nombre de fils) des différents signaux.

d'être indépendante de l'interconnexion (c'est son objectif principal). Il s'agit donc simplement de définir un brochage (un ensemble de fils, représentés sur la figure 5.8), et d'y associer un protocole de communication. L'interconnexion s'insère entre l'initiateur et le récepteur, mais du point de vue des composants, tout se passe comme s'il disposaient d'une liaison point à point entre eux. L'interconnexion est donc abstraite.

Il existe plusieurs déclinaisons de l'interface VCI correspondant à des besoins différents :

- **Périphérique.** C'est la version la plus simple, utilisée pour les périphériques qui n'ont pas besoin de requêtes avancées (rafales, etc.). Les requêtes et les réponses ne sont pas découplées (une seule poignée de main commune).
- **Basic.** Il s'agit de la version intermédiaire, utilisée pour les des communications plus avancées. Dans ce mode les rafales sont possibles et les requêtes et les réponses sont gérées de manières indépendantes (deux poignées de main).
- **Advanced.** C'est la version la plus complète (et la plus complexe) incluant la gestion des transactions séparées, des transactions dans le désordre (identifications des transactions), etc.

Une vue simplifiée du brochage de l'interface VCI *Advanced* est présentée sur la figure 5.8. Comme cela a été déjà introduit dans la section 5.2.1, il existe deux types d'interfaces, initiateur et récepteur. Des signaux séparés sont utilisés pour gérer les requêtes et les réponses. Voici une description des principaux signaux mis en jeu :

- **Requêtes**
  - `CMD_VAL` et `CMD_ACK` sont les fils en charge de la poignée de main entre l'initiateur est le récepteur. L'initiateur signifie au récepteur qu'une requête est valide en plaçant `CMD_VAL` à 1, et le récepteur indique qu'il est prêt à la recevoir en plaçant `CMD_ACK` à 1. Lorsque ces deux fils sont à 1, alors la requête est validée. Cela permet un fonctionnement asynchrone.
  - `CMD_ADDRESS` correspond à l'adresse de la requête.

- CMD_COMMAND indique le type de requête (lecture, écriture, etc.).
- CMD_WDATA correspond à la valeur (pour une écriture).
- CMD_EOP est un bit de fin de paquet (*End Of Packet*) pour la gestion des rafales.
- CMD_TID est un identifiant de la transaction utilisé dans le cas où l'initiateur peut envoyer différentes requêtes sans attendre les réponses (transactions séparées). Les réponses pouvant potentiellement arriver dans un ordre différent, ces fils permettent de les identifier et de les remettre dans l'ordre.

- **Réponses**
  - RSP_VAL et RSP_ACK implémente la poignée de main pour la réponse.
  - RSP_EOP indique la fin d'un paquet réponse (dans le cas d'une rafale).
  - RSP_RDATA correspond à la valeur (pour une lecture).
  - RSP_TID identifie la réponse (même identifiant que la requête).

Ainsi un composant compatible VCI devra disposer de ce brochage et du circuit permettant de contrôler les fils en fonction des besoins en communications du composant. La plupart des bus et des réseaux sur puce ne sont pas directement (nativement) compatibles avec une interface standard. Il est néanmoins possible (c'est en pratique très souvent le cas) d'ajouter des adaptateurs pour convertir les communications VCI en communications compatibles avec le NoC. Ainsi, et c'est d'ailleurs le but principal de cette interface, un composant peut être développé indépendamment de l'interconnexion qui sera utilisée dans SoC où il sera intégré.

5.3.3 Trafic des différents composants

Dans cette section nous présentons en détail les communications effectuées par les différents types de composants.

5.3.3.1 Accélérateurs matériels

Les ASIC sont dédiés pour une tâche précise, et ils manipulent en général de grande quantité de données. Leur fonctionnement simplifié est le suivant :

1. Lecture de blocs de données en mémoire .
2. Exécution d'un traitement sur ces données.
3. Réécriture en mémoire.

Le trafic produit est directement fonction de l'étape en cours et de l'algorithme exécuté. On peut le définir par une succession de phases avec un motif particulier d'accès à la mémoire, répété un grand nombre de fois comme cela a été montré dans [FRS04]. Reproduire le trafic de ce type de composant ne pose donc pas de problème pour son concepteur qui connaît en détails les différentes phases de fonctionnement et donc les différents motifs d'accès à la mémoire.

Les communications de ces composants sont très importantes en débit (jusqu'à plusieurs Go/s). De plus les traitements effectués sont souvent associés à des contraintes temps réel, c'est à dire que l'exécution ne doit pas dépasser un certain nombre de cycles d'horloge. Étant donné le temps de communication est plus important que le temps de calcul, il convient que ces composants disposent d'un accès privilégié au réseau d'interconnexion en bénéficiant par exemple d'une bande passage garantie.

5.3.3.2 Processeurs et caches

Les processeurs exécutent un code binaire stocké en mémoire. Ils sont en général associés à une mémoire cache (aussi simplement appelé cache), qui permet de garder des données près du processeur. Les accès à la mémoire, et donc les communications, sont alors effectués par le cache, c'est pourquoi je présente son fonctionnement ci-après.

Une mémoire cache est une petite image de la mémoire principale (quelques kilo-octets). Le code (les instructions à exécuter) et les données (opérandes des instructions) étant dans des parties différentes de la mémoire, il est avantageux de séparer le cache en deux, ou d'utiliser un cache pour le code (cache d'instruction) et un pour les données (cache de données). Cette mémoire permet de tirer parti des deux types de localités présentes dans la plupart des applications :

- **Localité spatiale** : Les accès à la mémoire ne sont pas aléatoirement répartis dans l'espace d'adressage, il est très fréquent que les données soient accédées de manière successive. Par exemple les accès au instructions d'un programme sont consécutifs, à l'exception des branchements et de sauts [119].
- **Localité temporelle** : Les données qui ont été accédées récemment ont de grandes chances d'être réutilisées. Lorsque l'on fait des calculs, le résultat d'un calcul a de fortes chances d'être réutilisé rapidement.

Un cache est organisé en M lignes contenant chacune N mots. La capacité du cache est donc $M \times N$ mots. Le principe de fonctionnement d'un cache est relativement simple. La mémoire est virtuellement segmentée en blocs de la taille d'une ligne de cache. Les lignes de cache sont associées à ces blocs comme cela sera expliqué plus bas. Lorsque le processeur a besoin d'une donnée, si la donnée est présente dans le cache, alors celle-ci est transmise au processeur en un cycle, sinon on a un défaut de cache (*cache miss* en anglais) et la ligne correspondant à l'adresse demandée doit être chargée depuis la mémoire principale avant que la donnée puisse être transmise au processeur. Il existe différentes méthodes pour associer les lignes du cache aux données de l'espace d'adressage complet :

- **Correspondance directe** (*Direct-mapped*). Dans ce cas on va simplement calculer l'adresse *modulo* le nombre de lignes de cache pour déterminer la ligne de cache correspondant à une adresse donnée. Cette technique est la plus simple, néanmoins elle provoque de nombreux défauts de cache. En effet, si on accède successivement à deux adresses différentes qui se projettent dans la même ligne de cache, alors même si d'autres lignes ne sont pas utilisées, la ligne en question va être remplacer et sont précédent contenu, qui a de forte chance d'être de nouveau demandé rapidement, sera perdu.
- **Cache associatif** (*set-associative*). Pour diminuer le nombre de conflits, il est possible de regrouper les lignes de cache pour former des ensembles de k lignes. On a alors M/k ensembles et on va calculer l'adresse *modulo* M/k pour obtenir l'ensemble associé à l'adresse donnée. Comme l'ensemble contient plusieurs lignes, il faut mettre en place une politique pour remplacer une ligne parmi les k de l'ensemble en cas de défaut de cache. il existe différents algorithmes pour faire ce choix : au hasard, la plus ancienne (LRU pour *Least Recently Used*), la moins fréquemment utilisée (LFU pour *Least Frequently Used*). Un cache est dit complètement associatif (*full-associative*) s'il n'y a qu'un seul ensemble. C'est alors au cache de décider où stocker chaque ligne. Un cache associatif a de meilleures performances qu'un cache à correspondance directe, néanmoins sa complexité est supérieure (il faut de la logique pour gérer les ensembles).

Concernant les écritures, il existe deux grandes stratégies lorsque la donnée à écrire est pré-

sente dans le cache :

- **Écriture décalée** (*write-back*). Une requête d'écriture du processeur entraîne une modification dans le cache (si la donnée est présente), mais pas de modification immédiate dans la mémoire principale. La mémoire principale sera mise à jour lorsque la ligne sera expulsée du cache. Ce système devient complexe dans les systèmes multiprocesseurs avec des espaces de mémoire partagés. En effet il faut mettre en place un mécanisme pour savoir si les données en mémoire principale sont valides.
- **Écriture directe** (*write-through*). Une requête d'écriture du processeur est modifier dans le cache *et* dans la mémoire principale. On utilise alors un tampon d'écritures postés (*write-buffer* en anglais) dans lequel les requêtes d'écritures sont stockées avant d'être envoyées sur le réseau d'interconnexion.

Dans le cas où la ligne n'est pas présente dans le cache, il existe deux stratégies principales :

- **Écriture attribuée** (*write-allocate*). La ligne est chargée dans le cache puis modifiée.
- **Écriture non attribuée** (*no write-allocate*). La ligne n'est pas chargée dans le cache, la donnée est simplement modifiée dans la mémoire centrale.

Dans le cas de systèmes multiprocesseur (comme les MPSoC par exemple), une nouvelle problématique apparaît, il faut gérer la *cohérence des caches*. En effet, comme les différents processeurs vont utiliser des espaces de mémoire partagés, il faut pouvoir s'assurer que les données lues sont valides. Il existe pour cela différentes approches :

- **Cache espion**. Si l'interconnexion est un support partagé (un bus typiquement), alors le cache voit passer toutes les requêtes de tous les composants. Il suffit pour cela qu'il dispose d'un *espion* qui écoute ce qu'il se passe sur le bus en permanence. Il peut alors détecter une écriture et mettre à jour la donnée dans son cache si elle est présente. Ce système est assez simple à mettre en place et fonctionne très bien, en revanche l'arrivée des réseaux sur puce, où il n'y a plus de support partagé par tous les composants, rend ce système impossible à utiliser.
- **Répertoire de cache**. Ceci implique la présence d'un répertoire centralisé contenant l'état de chaque ligne de la mémoire centrale (valide ou non) ainsi qu'une liste des processeurs ayant une copie de cette ligne. Ainsi à chaque requête d'écriture, le répertoire pourra envoyer des messages à chaque processeur maintenant une copie de la ligne dans leur cache afin qu'il se mette à jour. Ce système est complexe et le répertoire occupe un espace mémoire important, c'est pourquoi il n'est pas utilisé dans les SoC.
- **Solution logicielle**. Pour régler le problème de la cohérence des caches, on peut aussi laisser au programmeur la gestion du cache par l'intermédiaire d'une *mémoire brouillon* (*scratch-pad memory* en anglais). Le cache devient un espace d'adressage spécial de la mémoire, et c'est au programmeur de le gérer.
- **Suppression des problèmes de cohérence**. On peut enfin régler ce problème en spécifiant des morceaux de l'espace d'adressage *non cachable*. L'accès à cet espace sera fait sans passer par le cache. Dans un système multiprocesseur, si on déclare les espaces de mémoire partagés par au moins deux processeurs comme non-cachable, alors il n'y a plus de problème de cohérence des caches. Il est aussi bien entendu possible de supprimer les caches (c'est la solution retenue par Philips dans ses SoC jusqu'à présent).

Ces caractéristiques permettent de tirer des conclusions sur le trafic produit par un cache de processeur dans un SoC :

- **Lectures**. Les requêtes de lecture font toujours la taille d'une ligne de cache (mise à jour d'une ligne). Le temps entre deux lectures correspond à un temps pendant lequel le processeur travaille uniquement sur des données qui sont présentes dans le cache. On peut aussi distinguer deux flux : celui des données (cache de données) et celui des instructions (cache d'instructions).
- **Écriture**. Les requêtes en écriture sont de taille 1 mot dans le cas d'une politique d'écriture directe, et de la taille d'une ligne de cache dans le cas d'une politique d'écriture décalée. Si il y a un tampon d'écritures postées (*write-buffer*), alors celui-ci est périodiquement vidé et les requêtes peuvent être de taille variable.
- **Autres requêtes**. Pour l'accès aux parties de la mémoire déclarées "non-caché", ainsi que pour la gestion des requêtes de lecture/écriture atomique, certaines requêtes de 1 mot seront aussi effectuées. Si une politique de gestion de cohérence de cache est mise en place, alors des messages associés seront aussi envoyés.

L'étude des performances des caches est un domaine de recherche très actif (le lecteur intéressé est renvoyé vers [8, 77, 161, 7, 29] est les références incluses). Dans le cadre des systèmes embarqués, on préfère en général des solutions simples (cache à correspondance directe et écriture directe).

Le trafic d'un processeur est donc la somme de trois trafics et comme, en général, il ne dispose que d'une interface de communication, ces trois trafics sont entrelacés. Afin de garantir la consistance mémoire, il est nécessaire que l'ordre des lectures et des écritures soit préservé. Il est à noter que si le processeur n'a pas de cache, il effectue alors *en permanence* des accès mémoire (pour accéder aux instructions et aux données) car le temps d'accès à la mémoire est largement supérieur au temps de calcul. Le processeur sera, la majorité du temps, en attente des instructions et des opérandes, et les performances seront diminuées. Le cache de données peut être néanmoins supprimé, si l'on sait par exemple que les données sont accédées consécutivement (pas de localité temporelle). C'est souvent le cas des processeurs spécialisés comme les DSP par exemple.

5.4 Génération de trafic sur puce

Cette section présente un état de l'art centré sur la problématique que nous avons étudiée dans cette partie de la thèse, à savoir la génération de trafic dans les SoC pour l'évaluation de performance des NoC.

On peut distinguer deux grandes approches pour la génération de trafic sur puce. L'approche déterministe (présentée dans la section 5.4.1), qui consiste à chercher à reproduire exactement le trafic d'un composant, et l'approche stochastique, qui consiste à modéliser le trafic à l'aide de processus stochastique et à synthétiser des réalisations de ce processus pour générer du trafic (présentée dans la section 5.4.2).

5.4.1 Génération de trafic déterministe sur puce

Genko *et al.* ont développé un générateur de trafic capable de rejouer une trace constituée d'une suite de paquets ayant trois caractéristiques : temps d'injection, taille (nombre de mot), destination (adresse) [52]. Ceci est intégré dans un environnement d'émulation de réseaux sur puce dans des FPGA, la générateur de trafic est donc un composant matériel, et la trace doit être

stockée dans la mémoire du FPGA. La simulation est donc très rapide (typiquement 50 Mhz) et permet une évaluation de performance rapide des NoC. Pour cela, des récepteurs de trafic calculent des statistiques.

Loghi *et al.* ont proposé un environnement pour l'évaluation de performance des NoC dans un contexte MPSoC[93]. Dans ce travail, les processeurs sont simulés, il n'y a pas à proprement parler de génération de trafic, un code est exécuté sur les processeurs.

Mahadevan *et al* [98] ont proposé une méthode pour reproduire le trafic d'un processeur. Leur approche part d'une trace de trafic du processeur (associé à son cache) enregistré lors d'une simulation. Cette trace est ensuite convertie (compilée) vers un code binaire très simple constitué de 7 instructions seulement : lecture, écriture, rafale de lecture, rafale d'écriture, attente, saut et saut conditionnel. Le générateur de trafic est donc un processeur très simple pouvant exécuter ce jeu d'instruction. L'exécution de ce code génère le même trafic (à la gestion des sémaphores près) que si le processeur complet était simulé, et on gagne ainsi en temps de simulation (2 à 4 fois plus vite en fonction du nombre de processeurs). L'erreur (différence de nombre de cycle entre le générateur de trafic et la simulation originale) est très faible (inférieure à 2%). Cette approche est intéressante, néanmoins elle ne présente pas d'autre intérêt que l'accélération des simulations d'un facteur qui n'est pas déterminant. Ceci s'explique par le fait que la majeure partie du temps de simulation est passée dans la simulation de l'interconnexion (voir section 8.1). Aucune information n'est donnée sur la taille des programmes ainsi générés.

5.4.2 Génération de trafic aléatoire sur puce

Une grande proportion des travaux concernant l'évaluation de performance des réseaux sur puce est faite à l'aide de générateurs de trafic stochastiques [80, 155, 146, 125]. Cela vient du fait que les NoC sont encore dans une phase exploratoire, et évaluer leur comportement face à des sources aléatoires est un bon moyen pour guider des décisions à grande échelle (topologie, routage, mémorisation dans les commutateurs, etc.).

Lahiri *et al.* utilisent par exemple des processus IID suivant des lois marginales classiques (exponentielle, Normale, etc.) pour générer les tailles des transactions et les délais entre deux transactions [80]. Ces générateurs de trafic sont utilisés pour évaluer et comparer différentes interconnexions (bus, bus hiérarchiques et réseaux sur puce).

Wilkund *et al.* génèrent du trafic aléatoire pour évaluer les performance de SOCBUS [155], et Thid *et al.* font de même avec NOSTRUM [155]. Pestana *et al.* [125] définissent une distribution spatiale uniforme (chaque initiateur communique avec tous les récepteurs de manière équiprobable) et une distribution gaussienne des temps entre deux transactions. Cette charge est utilisée pour évaluer et comparer différents algorithmes de routage.

Thid *et al.* utilisent du trafic synthétique à longue mémoire dans un travail très récent [147].

Varatkar et Marculescu ont proposé une modélisation et une méthode de génération de trafic sur puce au niveau macro-bloc dans le cadre d'une application de décodage vidéo MPEG2 [151, 150]. Ils ont en particulier mis en évidence le fait que si dans un SoC, les différentes étapes du décodage (décodeur arithmétique, compensation de mouvement MC[2], transformée cosinus inverse IDCT[3] et dé-quantification) sont implémentées par différents composants, alors les communications entre certains de ces composants ont une caractéristique de longue mémoire. La longue mémoire ayant un impact important sur les performances de mémorisation (voir sec-

[2] *Motion Compensation*
[3] *Inverse Discrete Cosine Transform*

tion 2.2), ce résultat est d'une importance capitale pour le dimensionnement de ces *buffers*. Il a été obtenu partir d'une implémentation multithread de l'application MPEG2 décodant différents clips vidéo. Les traces des communications à l'entrée des étapes IDCT et MC ont été converties en débit agrégé et analysé à l'aide des méthodes R/S (décrite dans la section 2.7.1) et temps-variance (décrite dans la section 2.7.2) pour évaluer la présence de longue mémoire. Leur résultat est que ce trafic contient bien de la longue mémoire, avec un paramètre de Hurst situé, comme pour le trafic Internet, entre 0.7 et 0.8. Une méthode de synthèse de clip synthétiques est aussi proposée. Elle est est basée sur la méthode de l'inverse (voir section 3.1.1) pour la génération de réalisations de processus à longue mémoire non gaussiens. Ces travaux sont très intéressant, et c'est d'ailleurs eux qui ont motivés notre orientation vers la longue mémoire.

Conclusion

Nous avons dans ce chapitre introduit la notion de SoC ainsi que les problématiques associées à leur conception. En particulier, le composant en charge de faire transiter les communications entre les différentes IP est en train de subir un changement technologique majeure avec l'introduction de véritable réseaux sur puce, qui font passer aux premier plan de la conception de SoC l'interconnexion, et donc les méthodes permettant de la concevoir rapidement. La génération de trafic sur puce est un outil indispensable pour effectuer ce dimensionnement, et nous avons fait un état de l'art des différents travaux existant dans ce domaine qui constitue la problématique principale de cette partie de la thèse. Dans ces travaux, tous assez récents, on peut distinguer deux grandes approches : la génération de charge aléatoire simple sur le réseau pour évaluer différents algorithmes de routage, topologies, architectures de commutateurs, etc., et la génération par rejeu d'une trace issue d'une simulation du composant.

Afin de pouvoir évaluer ces différentes approches, et d'explorer cette problématique en détail, nous avons développé un environnement de génération de trafic complet et flexible, intégrant ces deux grandes approches, ainsi que la possibilité de générer du trafic à longue mémoire car cette propriété a été identifiée dans les communications entres les différents modules d'un décodeur MPEG2. La mise en place de cet environnement fait l'objet du chapitre suivant.

Chapitre 6
Contribution de la thèse à la génération de trafic sur puce

Ce chapitre présente notre approche de la génération de trafic sur puce, que nous avons mise en place et affinée tout au long de la thèse.

Nous nous intéressons à la génération de trafic sur puce, c'est à dire à reproduire artificiellement les communications d'un composant, sans avoir à le simuler. Le trafic des composants dédiés peut être déduit du fonctionnement interne du composant, alors que celui du processeur associé à une mémoire cache est beaucoup plus complexe à reproduire. Nous nous sommes donc concentré, dans nos travaux, sur la modélisation du trafic des processeurs. En effet, après avoir discuté avec la société ST Microelectronics, il est apparu qu'il y avait un besoin réel de ce type de modélisation, alors que le trafic des accélérateurs matériels était en pratique effectué par leurs concepteurs. Le modèle de génération de trafic de processeurs chez ST Microelectronics est en effet très simpliste (un taux de défauts de cache, correspondant à une charge constante), et ils étaient intéressés par avoir des modèles plus fins de ce type de trafic. Au départ de la thèse, il était prévu que nous travaillions sur des traces fournies par ST Microelectronics, issue d'émulation de SoC réel (en cours de finalisation pour fabrication). Nous n'avons pas pu au final travailler sur ces traces, c'est pourquoi nous avons effectué nous même des simulations dans un environnement de simulation universitaire (SoCLib [114]). Il se trouve de plus que cet environnement contient uniquement des processeurs généralistes (pas d'accélérateurs matériels, ni de processeurs spécialisés). Nos travaux se sont, pour ces raisons, limités à l'étude du trafic émis par un processeur associé à son cache. L'extension de cette étude aux trafics issus d'autres composants fait partie des perspectives de recherche.

L'organisation de ce chapitre est la suivante : la section 6.1 introduit une formalisation des communications au sein d'un SoC et la section 6.2 présente l'environnement de génération de trafic que nous avons mis en place. Un résumé des points clefs de notre environnement de génération de trafic est enfin proposé dans la section 6.3.

6.1 Modélisation du trafic sur puce

Cette section présente, dans un premier temps, une formalisation du trafic émis par un composant intégré dans un SoC (section 6.1.1) puis, dans un second temps, explique comment ce formalisme permet d'effectuer une modélisation de ce trafic (section 6.1.2). L'aspect multiphase du trafic est enfin discuté dans la section 6.1.3.

6.1.1 Formalisme

On peut représenter le trafic émis par un composant au niveau de l'interface standard de communication par une suite de *transactions*. Une transaction correspond à un couple (*paquet requête*, *paquet réponse*) et la $k^{\text{ième}}$transaction $T(k)$ est définie par un vecteur à six composantes $T(k) = \{A(k), C(k), S(k), D(k), R(k), L(k)\}$. La figure 6.1 explicite ce formalisme et voici la définition de chacune des composantes :

- $A(k)$. **Adresse de destination** de la transaction. Dans le cas d'une requête multiple, les adresses sont successives, on gardera donc la première adresse. Cette information est capitale puisque l'adresse détermine la destination du paquet, et donc le parcours du paquet dans le réseau.
- $C(k)$. **Commande** de la transaction (lecture, écriture, etc.). Cette information est utile car certains composants peuvent gérer différemment les lectures et les écritures (un cache par exemple, voir section 5.3.3.2).
- $S(k)$. **Taille du paquet requête** (en nombre de mots). Cette information permet de caractériser la charge de la transaction pour le réseau.
- $R(k)$. **Taille du paquet réponse** (en nombre de mots) . Cette taille peut être différente de celle de la requête, cela dépend du niveau de protocole de l'interface standard utilisée.
- $D(k)$. **Délai** entre la réception de la réponse et le début de la transaction suivante (en nombre de cycles). Cette information, couplée à la latence $L(k)$, permet de placer dans le temps les paquets requête et réponse de chaque transaction. Ce délai peut être négatif, si le composant peut envoyer une requête sans attendre la réception de la réponse précédente (transactions séparées ou *split-transactions*).
- $L(k)$. **Latence**, c'est à dire le temps écoulé entre le début de la $k^{\text{ième}}$requête et le début de la réponse associée (en nombre de cycles). Cela correspond au temps d'aller retour dans le réseau, et $L(k)$ peut être utilisée pour caractériser la latence du réseau.

Cette séquence de vecteurs caractérise précisément les communications d'un composant d'un point de vue trafic, et va nous permettre de générer du trafic sur puce. Les autres informations de l'interface standard (identification de paquet, de transaction, etc.) ne sont pas utiles pour caractériser ce trafic en vue d'évaluer la performance du réseau.

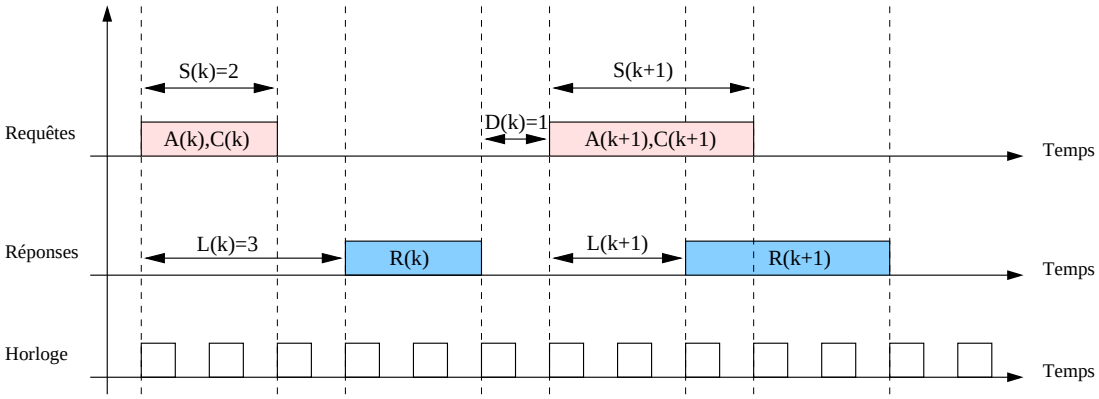


Fig. 6.1: Formalisation des communications d'un composant d'un SoC au niveau de l'interface standard. La $k^{\text{ième}}$et la $(k+1)^{\text{ième}}$transactions sont représentées.

Ce formalisme est valable pour toute interface standard, et la séquence de transactions peut facilement être extraite des variations de quelques signaux comme cela sera expliqué dans la section 6.2.5.

Nous allons par la suite considérer donc que la séquence de transactions $T(k)$ caractérise le trafic d'un composant particulier. En fonction de la nature du composant dont on souhaite caractériser le trafic, le formalisme peut être simplifié :

- **Taille des requêtes/réponses**. Il peut arriver que le protocole de communication de l'interface standard force les réponses à avoir la même taille que les requêtes (ce sera le cas dans nos expérimentations). Dans ce cas, on peut se passer de considérer la taille des réponses $R(k)$ puisque $\forall k, R(k) = S(k)$.
- **Latence**. La latence des transactions $L(k)$ sert à caractériser le placement temporel des réponses. Si on utilise un générateur de trafic en remplacement d'un composant, mais que l'on maintient la simulation des composants à qui sont adressées les requêtes (les mémoires et les périphériques), alors il n'est pas préférable, pour le générateur de trafic, de considérer $L(k)$, cette information sera simulée par le réseau et le destinataire. D'autre part, cette séquence va changer lorsque l'on va changer l'interconnexion (la latence du réseau va changer), et comme nous souhaitons avoir une caractérisation du trafic *indépendante* du réseau, il est nécessaire de ne pas considérer $L(k)$ dans la formalisation du trafic, mais plutôt utiliser $D(k)$ pour placer les requêtes dans le temps. La latence $L(k)$ sera en revanche utilisée comme un indicateur dynamique de l'état du réseau, car elle est directement lié à la congestion au sein de celui-ci.

Si l'on fait ces deux simplifications, le trafic est alors une séquence de transactions $T(k) = \{A(k), C(k), S(k), D(k)\}$. Nous allons par la suite considérer cette séquence car les deux simplifications indiquées sont valables pour nos expérimentations.

A partir de cette séquence, on peut introduire le débit agrégé dans des fenêtres temporelles de taille Δ noté $W_\Delta(i)$ correspondant au nombre de mots transmis entre les cycles $i\Delta$ et $(i+1)\Delta$ comme cela a été déjà introduit dans la première partie de la thèse (sections 1.2.6, 2.3 et 4.1). L'information contenue dans le débit agrégé rassemble la taille des requêtes et le placement temporel de celles-ci, on peut l'utiliser à la place de $D(k)$ et $S(k)$. En revanche, $W_\Delta(i)$ ne donne aucune information sur l'adresse de destination et sur les commandes.

6.1.2 Modélisation

Nous allons considérer le trafic d'un composant comme une séquence (ordonnée) de transaction $T(k) = (A(k), C(k), S(k), D(k))$ comme cela a été introduit dans la section précédente. Modéliser le trafic, c'est choisir un modèle pour $T(k)$.

Notre approche de la modélisation de trafic est orientée traces, c'est à dire que nous n'allons pas chercher à modéliser le comportement d'un composant pour en modéliser les communications, mais plutôt travailler à partir d'une trace de référence, de laquelle nous allons extraire une séquence $T(k)$. Cette séquence de référence sera alors modélisée et différentes possibilités s'offrent alors à nous :

1. **Rejeu**. On peut choisir de *rejouer* la séquence de référence à l'identique. Il n'y a pas dans ce cas à proprement parler de modélisation, mais cela permet de valider la génération de trafic.
2. **Vecteur aléatoire indépendant**. On peut considérer chaque composante du vecteur $T(k)$,

de manière indépendante, comme la réalisation d'un processus stochastique. On cherche alors par la modélisation à capturer les propriétés statistiques de chacune des composantes sans considérer leurs corrélations.

3. **Vecteur aléatoire**. On peut considérer $T(k)$ comme un vecteur aléatoire dont les composantes ne sont pas indépendantes, et chercher non-seulement à caractériser les propriétés statistiques de chaque composantes, mais aussi leurs corrélations.
4. **Approche hybride**. Les trois approches précédentes sont des approches *boîte-noire*, c'est à dire qu'on analyse la série de donnée à notre disposition sans avoir d'*a priori* sur ces données. On peut, au contraire, adopter une approche hybride tenant compte des caractéristiques du composant dont on cherche à caractériser le trafic. Par exemple, si on modélise le trafic d'un cache, alors on sait, de par la nature du composant, que les requêtes de lecture ne peuvent pas avoir une taille arbitraire.

Nous avons aussi, à partir de la définition du débit agrégé, défini deux niveaux de précision de la modélisation :

- **Niveau délai**. Les séquences de délai et de taille des transactions sont utilisées dans la modélisation.
- **Niveau débit**. On utilise la séquence de débit agrégé afin de caractériser le délai et les tailles des transactions. Dans ce cas, on perd l'information précise du placement temporel et de la taille de chaque transaction, on a une vision à plus gros grains du trafic.

La formalisation introduite dans cette section est la base de notre environnement de génération de trafic, et elle sera utilisé constamment dans les chapitres précédents. Aussi simple et intuitive que cette formalisation puisse paraître, c'est à notre connaissance la première fois que le trafic sur puce est ainsi défini afin de pouvoir être modélisé.

6.1.3 Phases dans le trafic

Dans le cas où l'on désire modéliser par des processus stochastiques stationnaires la séquence de transactions (ou une composante de celle-ci), il faut alors s'assurer de la stationnarité de la séquence pour que la modélisation ait un sens (voir section 1.2.4). Nous avons observé, à partir de nos premières simulations, qu'au contraire le trafic émis par un processeur et son cache n'est pas stationnaire. On voit se dégager des *phases* dans le trafic. Nous avons alors décidé de prendre en compte cette caractéristique du trafic afin de pouvoir effectuer une modélisation par des processus stochastiques stationnaires qui ait du sens. Ceci est illustré sur la figure 6.2, qui montre une trace de débit agrégé d'un processeur exécutant une application de décodage audio (MP3). On voit clairement apparaître sur cette figure différentes phases, c'est-à-dire des morceaux de la trace qui n'ont pas du tout le même comportement. Si on effectue une modélisation à base de processus stochastiques stationnaires sur la trace complète, le résultat n'aura pas de sens puisque clairement, la trace n'est pas stationnaire.

Nous avons donc introduit le concept de *phase de trafic*, qui va nous permettre de contourner cette difficulté. On définit une *phase* comme un morceau de la séquence de transaction compris entre deux indices i et j. Un phase est donc caractérisée par un couple (i, j). Étant donné que les mêmes traitements sont souvent effectués plusieurs fois au cours de l'exécution d'un algorithme sur un processeur, il y a une grande chance qu'une phase apparaisse plusieurs fois dans la trace et une phase devient alors une suite de couple d'indices : $\{(i_1, j_1), (i_2, j_2), .., (i_n, j_n)\}$, correspondant aux n apparitions de la phase dans la trace de trafic. Décomposer une trace en morceaux stationnaires est un problème très complexe que nous n'avons pas cherché à résoudre théoriquement.

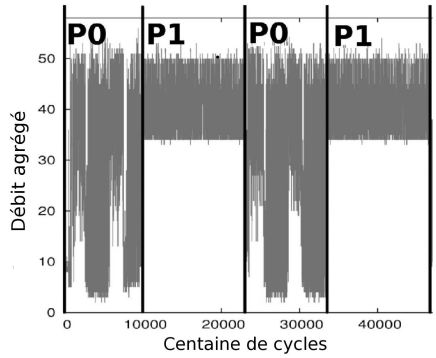


Fig. 6.2: Trace de débit agrégé d'un processeur exécutant une application de décodage audio MP3.

Nous avons opté pour une approche pragmatique, qui sera détaillée dans la section 6.2.3.

Tenir compte de l'aspect multiphase du trafic, implique d'une part qu'une modélisation soit effectuée sur chaque phase, et d'autre part que le générateur de trafic gère le basculement d'une phase à l'autre. Cela sera expliqué en détails dans la section 6.2.

6.1.4 Conclusion

La modélisation de trafic sur puce est une problématique complexe et encore assez peu étudiée. Il s'agit de faire des choix (précision, type de modélisation), qui dépendent de l'utilisation qui sera faite de la modélisation. Nous pensons que, par analogie avec la conception de puce (voir section 5.1.4.3), la génération de trafic doit être incrémentale. Au fur et à mesure que la conception du SoC avance, l'interconnexion doit être testée sous des charges de trafic de plus en plus proches des charges de trafic réelles finales. Notre objectif a été de définir un environnement de génération de trafic assez complet et flexible permettant au concepteur d'effectuer cette génération de trafic incrémentale. Cet environnement fait l'objet la section suivante.

6.2 Environnement de génération de trafic

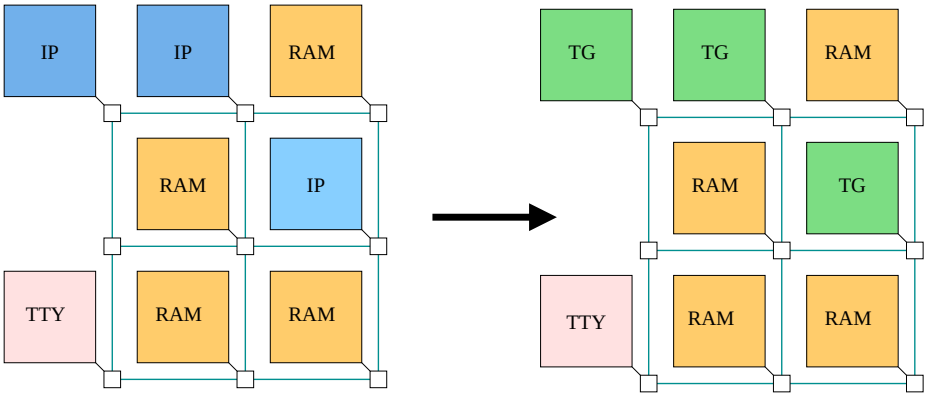


Fig. 6.3: Utilisation des générateur de trafic dans les SoC.

Cette section présente l'environnement de génération de trafic que nous avons mis en place

tout au long de la thèse. L'objectif principal de cet environnement est de remplacer les composants (IP) par des générateurs de trafic (TG pour *Traffic Generator*) comme cela est illustré sur la figure 6.3. Cela permet entre autre d'accélérer les simulations et permet de procéder à l'exploration architecturale de l'interconnexion de manière plus rapide et plus flexible.

Un tel générateur de trafic peut aussi être utile pour régler des problèmes de propriété intellectuelle. En effet, comme cela a été discuté dans la section 5.1.4.3, les concepteurs de SoC vont, de plus en plus, utiliser des composants (IP) achetés auprès de sociétés spécalisées qui doivent fournir différents modèles de simulation du composants à différents niveaux. Ceci pose un problème de propriété intellectuelle si ces modèles contiennent des informations sensibles. La diffusion d'un générateur de trafic paramétrable à la place du composant peut permettre de résoudre ce problème.

Nous avons pour cela mis en place un flot global présenté sur la figure 6.4. Ce flot comporte trois étapes principales : l'obtention d'une trace de référence, l'analyse de cette trace de référence et la construction de générateurs de trafic adaptés et enfin l'utilisation de ces générateurs pour effectuer l'exploration architecturale du réseau d'interconnexion. Chaque étape est décrite en détail dans les sections suivantes. L' objectif de ce flot est d'offrir aux concepteurs un outil de génération de trafic sur puce flexible, adapté pour traiter un maximum de situations, mais restant générique. Un des points clés de ce flot est la segmentation en phases du trafic (section 6.2.3), une autre étape décisive est de garantir que l'on peut enregistrer une trace de référence avec une interconnexion donnée arbitraire, et utiliser cette trace pour générer du trafic réaliste sur n'importe quelle interconnexion (ce point est discuté dans la section 6.2.5).

La mise en place de cette environnement ainsi que des résultats préliminaires ont été publiés dans les actes de la conférence *ASAP* 2006 [SFR06b]. Cela a aussi fait l'objet d'un rapport de recherche [SFR06c].

6.2.1 Trace de référence

La première étape du flot (voir figure 6.4) est la collecte, par simulation, de la trace de référence, sur laquelle les analyses seront effectuées. Lors de cette simulation, il n'est pas nécessaire de simuler une interconnexion réelle, on peut soit utiliser une interconnexion générique, soit connecter directement le processeur à une mémoire contenant toutes les informations nécessaires (codes et données). La simulation est ainsi plus rapide que si l'on simule l'interconnexion, et nous montrerons plus loin que l'on peut, au prix d'une erreur minime, utiliser les résultats de cette simulation pour générer du trafic sur n'importe quelle interconnexion.

La simulation est effectuée dans l'environnement SoCLib basée sur le moteur de simulation SystemC (décrit dans la section 7.1) et on obtient en sortie de cette simulation une trace au format VCD (*Value Change Dump*) qui enregistre dans un format textuel les variations de certains signaux de l'interface standard (SoCLib utilise VCI).

6.2.2 Parseur

La trace VCD obtenue lors de la simulation de la référence contient les variations des signaux VCI suivants (voir section 5.3.2) :

- **Requêtes**
  - `CMD_VAL` : poignée de main.
  - `CMD_ACK` : poignée de main, identifie le début du paquet requête.

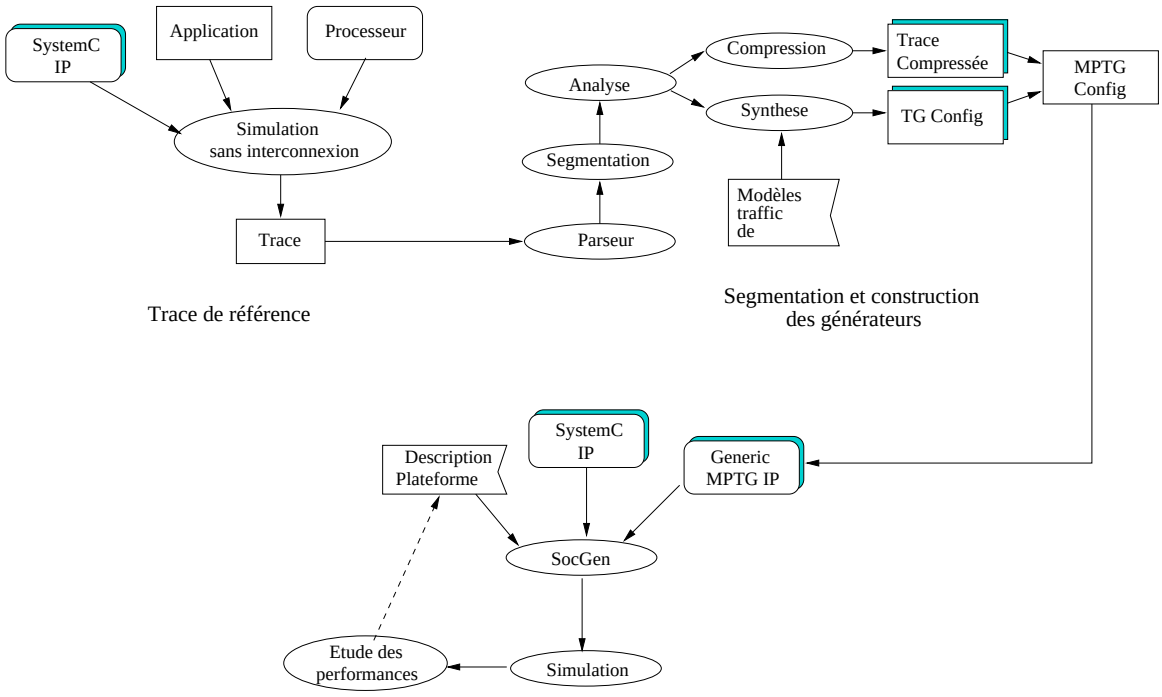


Fig. 6.4: Flot de génération de trafic multiphase et d'évaluation de performance de réseaux sur puce mis en place au cours de la thèse.

- `CMD_ADDRESS` : adresse de destination.
- `CMD_COMMAND` : commande.
- `CMD_EOP` : permet de déterminer la taille du paquet requête.

- **Réponses**
  - `RSP_VAL` : poignée de main.
  - `RSP_ACK` : poignée de main, identifie le début du paquet réponse.
  - `RSP_EOP` : permet de déterminer la taille du paquet réponse ainsi que le temps de fin de transaction.

A partir des variations de ces huit signaux, on peut extraire la séquence $T(k) = (A(k), C(k), S(k), D(k))$ des transactions introduite dans la section 6.1.2 ainsi que la latence de chaque transaction $L(k)$ et le débit agrégé $W_\Delta(i)$. Pour cela, nous avons écrit un analyseur syntaxique (*parser* en anglais) du format VCD, écrit en langage C et utilisant les outils de génération de *parser* FLEX et BISON [89]. Nous avons, à partir des spécifications du format VCD, décrit la grammaire de ce format, et à partir de la connaissance du protocole de communication de l'interface standard VCI, il est possible d'extraire toutes les informations précitées. Ce *parser* est disponible sous licence GPL.

6.2.3 Segmentation

Comme cela a été introduit dans la section 6.1.3, nous devons, avant de faire une analyse des données, segmenter la séquence $T(k)$ en phases stationnaires. Nous avons pendant une grande partie de la thèse effectué ce travail manuellement (à l'œil sur des graphiques représentant l'évo-

lution des différentes composantes de $T(k)$). Nous avons ensuite, fort de cette expérience, cherché à automatiser cette étape, et pour cela nous avons adopté une méthode inspirée des travaux de Brad Calder *et al.* concernant l'identification de phases de programmes (section 6.2.3.1). Notre adaptation de ces travaux dans le cas de la segmentation de trafic sur puce est décrite dans la section 6.2.3.2.

6.2.3.1 L'approche de Calder *et al.*

Les travaux de Brad Calder *et al.* se situent dans le domaine de l'évaluation de performance d'architecture de processeurs haute performance. Dans ce domaine, on utilise un simulateur d'instruction (ISS pour *Instruction Set Simulator*). Afin d'évaluer la performance du processeur, on le fait exécuter différents programmes contenant chacun plusieurs milliards d'instructions. Cette étape est très lente, et l'idée de Calder *et al.* est, dans le but d'accélérer ces simulations, d'identifier des *phases* dans les programmes, les phases étant des morceaux du flot d'instructions. Chaque phase identifiée est alors simulée une seule fois, et la performance du processeur exécutant l'application complète est directement déduite des résultats de simulation de chaque phases et de l'alternance des phases dans le programme. Cette technique, appelé SIMPOINT permet d'obtenir des gains très importants en temps de simulation, pour une erreur relativement faible. Le lecteur intéressé par cette technique est renvoyé au site WEB de SIMPOINT [43] ainsi qu'aux articles [138, 58].

Concrètement, la trace (ici le flot d'instructions) est tout d'abord segmentée en blocs consécutifs de taille fixe sans recouvrement (la taille est de l'ordre de quelques millions d'instructions). Sur chaque bloc i un vecteur V_i est défini contenant la fréquence d'apparition des *blocs de base* (*basic blocs* en anglais) du code (un bloc de base est un morceau de code assembleur séquentiel, sans saut ni branchement). Ce vecteur est ensuite ramené, par une technique de réduction, à un nombre de dimensions acceptable (une quinzaine), et l'algorithme de classification *k-means* [97] est alors utilisé pour regrouper les blocs dans des classes, qui seront les phases du programmes. On s'attend à ce que il y ait un grand nombre de blocs adjacents appartenant à la même phase.

L'algorithme *k-means* recherche, étant donné un ensemble de points d'un espace vectoriel de dimension n et un nombre de phases k, k *centres* (points de l'espace vectoriel). Chaque point de l'ensemble initial se verra affecter le centre dont il est le plus proche (au sens d'une distance qu'il faut définir, voir plus bas). Initialement, les centres sont choisis aléatoirement parmi les points de l'ensemble, puis ils sont itérativement déplacés afin de minimiser la somme des distances des points à leurs centres associés. L'algorithme termine soit au bout d'un nombre d'itérations fixé par l'utilisateur, soit lorsque la somme des distances des points à leurs centres associés est inférieure à un seuil fixé par l'utilisateur. On obtient ainsi une classification des points. D'un point de vue théorique, il est important de préciser que l'algorithme fait l'hypothèse que les données sont issues d'une mixture de points répartis de manière Gaussienne autour des k centres. Différentes distances peuvent être utilisées pour évaluer la distance entre deux points M et N d'un espace vectoriel de dimension n, en voici une liste non-exhaustive, une bonne revue de ces distances est disponible dans [15] :

- **Distance de manhattan**. $D_{man} = \sum_{i=1}^{n} |N(i) - M(i)|$
- **Distance Euclidienne**. $D_{euclid} = \sqrt{\sum_{i=1}^{n} |N(i) - M(i)|^2}$

- **Distance de Minkowski**. $D_{min} = \left(\sum_{i=1}^{n} |N(i) - M(i)|^p\right)^{1/p}$
- **Distance infinie**. $D_{inf} = max_{i=1,..,n}|N(i) - M(i)|$
- **Divergence de Kullbach-Leiber**. $D_{kl} = \sum_{i=1}^{n} M(i) \log \frac{M(i)}{N(i)}$. Elle est utilisé lorsque M et N caractérisent des distributions de probabilité (fréquences d'apparitions). Elle n'est pas symétrique, donc on utilise en général la moyenne entre la distance entre M et N et la distance entre N et M.

L'algorithme *k-means* demande que l'on fixe le nombre de phases, mais en général on ne connaît pas *a priori* le nombre qui donnera les meilleurs résultats. Il est néanmoins possible d'exécuter l'algorithme pour différentes valeurs de k et de sélectionner la "meilleure" classification. Pour comparer les classifications entre-elles, il existe différentes techniques. Celle utilisée par Calder *et al.* est le critère d'information bayésien [97] (BIC pour *Bayesian Information Criteria*). Ce critère utilise une fonction de vraisemblance L, caractérisant l'adaptation du modèle (ici une classification en k phases) à l'ensemble de données, mais pénalise les modèles ayant beaucoup de paramètres. Ainsi, plus le nombre de phases sera élevé, plus le modèle sera pénalisé, et donc plus l'adéquation du modèle aux données devra être forte. Sa définition est la suivante pour un modèle M à p paramètres et un ensemble de N échantillons noté X :

$$BIC(M) = \log L(X|M) - p\log(N) \tag{6.1}$$

Dans le cas spécifique de l'algorithme *k-means*, une méthode de calcul du BIC est donnée dans [124]. C'est cette méthode qui est utilisée par Calder *et al.*

6.2.3.2 Notre approche de la segmentation

Nous avons utilisé une approche similaire, même si les données sur lesquelles nous travaillons, ainsi que l'utilisation faite de la segmentation sont très différentes (voir section 6.1.3). En effet, Calder *et al.* cherche à caractériser le temps d'exécution d'un code sur une architecture de processeur, c'est pourquoi le critère guidant la segmentation (la fréquence d'apparition de morceau de code), permet de regrouper les traitements similaires. Nous avons un autre objectif, nous souhaitons identifier des périodes où le processeurs émet un trafic raisonnablement stationnaire, c'est à dire des morceaux de la séquence de transactions qui pourront être raisonnablement modélisés par des processus stochastique stationnaires. En pratique, les traitements ayant une influence très forte sur les communications, il y de fortes chances pour que les phases de trafic correspondent dans une certaine mesure aux phases des programmes.

Concrètement, la trace $T(k)$ est dans un premier temps coupée en blocs consécutifs disjoints. Pour chacun de ces blocs, nous définissons un **vecteur de bloc**, noté BV pour *Bloc Vector*, constitué d'une ou plusieurs statistiques (moyenne, variance, distribution de probabilité, covariance), calculées sur une ou plusieurs composantes de la séquence de transaction $T(k)$. Nous utilisons ensuite l'algorithme *k-means* avec ce vecteur pour associer à chaque bloc une phase. Le BIC est calculé (comme dans[124]) pour différentes valeurs de k afin de pouvoir déterminer le nombre de phases. A la fin de cette étape on a une liste de k phases, c'est à dire une liste de couples d'indices identifiant les morceaux de la trace qui appartiennent à chacune des phases. La pertinence de cette technique de segmentation sera discutée à partir de résultats expérimentaux dans la section 8.3. Ces résultats ainsi que la définition de la méthode ont fait l'objet d'une publication dans la conférence *CODES-ISSS* 2006 [SFR06a].

Il est à noter que cette segmentation a un intérêt en soi, car elle donne une information sur l'alternance de différentes phases aux caractéristiques statistiques différentes, c'est à dire différentes phases qui vont avoir un impact différent sur le réseau d'interconnexion.

6.2.4 Analyse des phases

Ayant identifié les phases dans la séquence de transactions originale, on peut maintenant, pour chaque phase, choisir une modélisation. Les choix que cela implique sont pour l'instant laissés au concepteur. Dans la modélisation, nous considérerons deux sous-ensembles des composantes de la séquence de transactions $T(k)$ comme indépendantes : le *contenu* des transactions d'une part, c'est à dire les adresses, commandes et tailles $(A(k), C(k), S(k))$ et *le placement temporel* des transactions d'autre part, c'est à dire les délais $(D(k))$. Ceci revient à séparer le contenu d'une transaction, du temps qui s'écoule entre deux transactions. Cette hypothèse est justifiée intuitivement par le fait qu'il n'y a pas, *a priori*, de raison que le contenu d'une transaction soit fortement corrélé avec le temps s'écoulant entre deux transactions successives. Des résultats expérimentaux en faveur de ce postulat seront présentés dans la section 8.3.2.

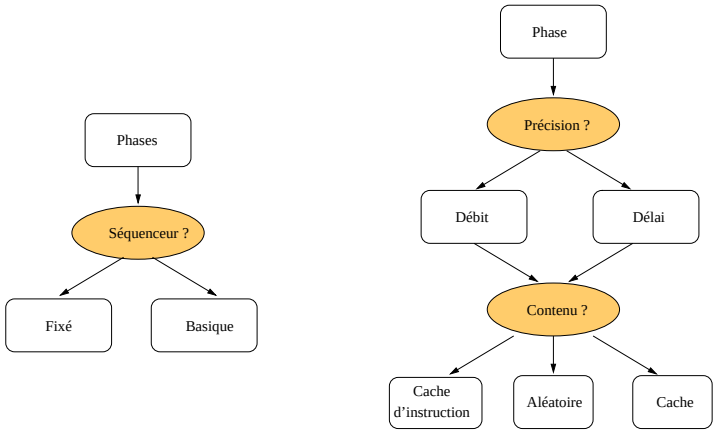


Fig. 6.5: Diagramme des choix (bulles rondes) que doit faire le concepteur pour dériver un générateur de trafic à partir d'une trace segmentée en phases.

Les choix que le concepteur doit faire à cette étape du flot sont décrits sur la figure 6.5, et ils sont détaillés ci-après :

1. **Précision**. Le concepteur doit tout d'abord choisir entre une précision au niveau transaction (délai et taille des transactions) ou niveau débit agrégé (on utilise le débit et on ne considère plus les délais). Cela dépend du contexte et de l'objectif de la génération de trafic. En utilisant le débit agrégé, on perd l'information précise des délais entre transactions, mais la charge induite (à une échelle de temps supérieure à la taille Δ de la fenêtre) est respectée. Si on choisit la précision débit agrégé, alors les deux sous-ensembles considérés indépendants seront : adresses, commandes et taille $(A(k), C(k), S(k))$ d'une part et le débit agrégé $(W_\Delta(i))$ d'autre part.
2. **Modélisation du contenu des transactions**. Une fois que la précision a été décidée, il faut définir la modélisation du contenu (adresse, commande et taille). Nous avons défini différents types de modélisation adaptés à différentes situations :
   - **Aléatoire**. Dans ce mode chaque composante (Adresse, commande, délai et taille ou débit agrégé) est modélisée indépendamment des autres. Ceci peut être utilisé pour si-

muler une charge aléatoire sur le réseau ou pour rejouer simplement la trace de référence. Les différentes modélisations aléatoires possibles sont détaillées dans la section suivante.

- **Cache**. Ce mode est spécifique à un cache mixte instruction et donnée, ou à un cache de données. Dans ce mode la taille des lectures est constante (égale à la taille d'une ligne de cache), la taille des écritures est modélisée par un des modèles présentés dans la section suivante. Pour les adresses et les commandes, une table de probabilité à deux niveaux est calculée comme cela est expliqué dans la section suivante.
- **Cache d'instruction**. Ce mode est spécifique à un cache d'instruction. Il en contient les spécificités, c'est à dire que les accès sont uniquement des lectures de la taille d'une ligne de cache.

3. **Modélisation du placement temporel des transactions**. Suivant la précision choisie, les délais ($D(k)$) ou le débit agrégé ($W_\Delta(k)$) sont modélisés (voir section suivante).
4. **Modélisation de la durée des phases**. Une phase pouvant apparaître plusieurs fois dans la trace, il faut caractériser la taille, en nombre de transactions, de chaque phase. Cette information est modélisée comme indiqué dans la section suivante.
5. **Ordre d'apparition des phases**. Cette étape concerne la configuration du séquenceur en charge de choisir la séquence des phases. Nous avons pour cela défini deux politiques :
   - **Fixé**. L'ordre des phases est dans ce cas stocké dans un fichier, on peut utiliser ce mode pour rejouer la même séquence de phase que celle de la trace originale.
   - **Basique**. Les phases sont jouées dans les unes après les autres, dans l'ordre de leur numéro.

   On pourrait aussi considérer la séquence des phases comme la réalisation d'un processus stochastique et la modéliser comme n'importe quelle série. Nous n'avons néanmoins pas implémenté cette fonctionnalité dans notre environnement. Cela ne présenterait aucune difficulté, mais nous n'en avons pas ressenti l'utilité.

6.2.4.1 Modélisation d'une série

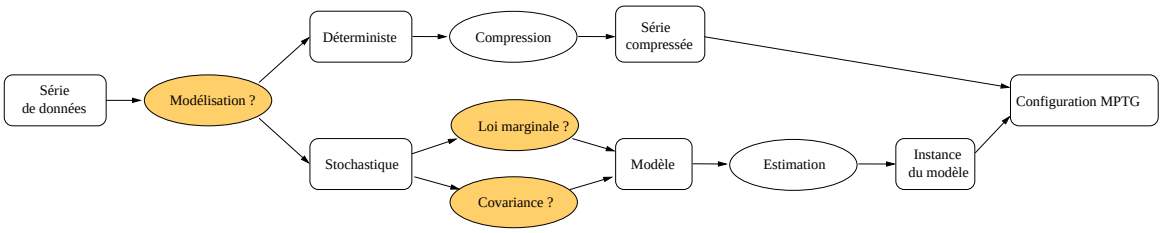


Fig. 6.6: Modélisation d'une série de données. Les bulles rondes grisées correspondent à des choix et les bulles rondes blanches à des actions automatisées.

Comme cela a été montré dans la section précédente, en fonction des choix du concepteur, une ou plusieurs séries de données vont devoir être modélisées (débit, taille, délai, etc.). Nous détaillons ici les choix des modèles possibles que nous avons implémentés pour modéliser une série $X(k)$. Ces choix ainsi que les actions à effectuer sont décrits sur la figure6.6. On peut distinguer deux grands types de modélisation, comme cela a déjà été noté dans la section 5.4 :

- **Rejeu**. On peut vouloir rejouer exactement la série originale, dans ce cas elle est stockée dans un fichier puis compressée de manière standard en utilisant l'algorithme de com-

pression par bloc BZ-2. Le taux de compression est discuté dans la section 8.2.2.

- **Processus stochastiques**. On peut aussi considérer la série comme la réalisation d'un processus stochastique. On doit alors, dans un premier temps, choisir un modèle de processus stochastique, puis dans un second temps utiliser une procédure d'estimation des paramètres du modèle sur la série originale (ces méthodes sont décrites dans la section 1.4.2, page 26). Nous allons ici utiliser les travaux décrits dans la première partie de la thèse, c'est à dire que nous allons effectuer une modélisation conjointe des statistiques d'ordre un (loi marginale) et des statistiques d'ordre deux (covariance). Le tableau 6.1 liste les formes de lois marginales et les formes de covariances disponibles dans notre environnement. Le modèle d'une série est le couple (forme de loi marginale, forme de covariance) et les paramètres sont l'union des paramètres de la forme de loi marginale et de la forme de covariance. Toutes les combinaisons (loi marginale/covariance) sont possibles. Pour chaque modèle, nous disposons de procédures d'estimation des paramètres et de génération de réalisations synthétiques.

Il est important de noter qu'à notre connaissance aucune modélisation de ce type n'a été effectué sur du trafic ayant lieu sur puce, tous les travaux décrits dans l'état de l'art (section 5.4.2) utilisent des processus stochastiques pour générer du trafic, mais ils ne modélisent pas le trafic, c'est à dire que les paramètres des processus utilisés sont fixés manuellement. Nous proposons, au contraire, une procédure d'estimation des paramètres ce qui implique que le trafic généré sera statistiquement proche de la trace originale. Notre travail est donc original, même si les méthodes et les outils d'estimations ainsi que les méthodes génération utilisées sont des outils relativement standards. Nous avons de plus l'originalité d'utiliser des processus stochastiques à longue mémoire (voir section 2).

Loi marginale	Paramètres
Normal	μ,σ^2
Exponentielle	μ
Gamma	α,β
Uniforme	a,b
Pareto	α,β
Empirique	$\mathbb{P}(X=i)^a$

(a)

Covariance	Paramètres
IID	Aucun
FGN	H
FARIMA$(1,d,1)$	ϕ,d,θ

(b)

Tab. 6.1: Liste des formes de loi marginale (a) et des formes de covariance (b) de processus stochastiques disponibles dans notre environnement de génération de trafic.

[a] $i \in [min(X), max(X)]$

Le tableau 6.1 liste les modèles que nous avons retenus pour modéliser les trafic sur puce, et afin de convaincre le lecteur que chaque processus est utile, le tableau 6.2 justifie chaque forme de loi marginale et de covariance en explicitant dans quels cas elles peuvent être utilisées.

La procédure d'estimation des paramètres a déjà été décrite dans la première partie de ce manuscrit, l'estimation des paramètres des lois marginales est basé sur la méthode du maximum de vraisemblance (voir section 1.4.2), l'estimation du paramètre de longue mémoire est effectué par la méthode des ondelettes (voir section 2.7.3), et l'estimation des paramètres d'un FARIMA est décrite dans la section 4.1.4.

A l'issue de cette étape, pour chacune des phases identifiées, nous disposons d'un modèle de génération de trafic contenant le type de précision, le type de génération de contenu des tran-

Lois marginales	
Normale	C'est une loi marginale très classique et très utilisée. Elle permet de bien représenter des données réparties symétriquement autour d'une moyenne. Le débit fortement agrégé suit en général une loi bien approximée par une loi Normale (de part le théorème de la limite centrale, voir section 1.1.2).
Exponentielle	Cette loi est très utilisée pour caractériser des temps entre deux événements (comme les arrivées de clients dans une file d'attente par exemple). Elle est donc adaptée à la modélisation du délai.
Gamma	La famille des lois Gamma est intéressante car elle balaye de manière continue les lois entre la loi exponentielle et la loi Normale (voir section 4.1).
Uniforme	La loi uniforme est classiquement utilisée et utile dans de nombreux cas. Par exemple pour tester un réseau sous une charge spatialement aléatoire, on va utiliser une loi uniforme pour l'adresse de destination.
Pareto	Cette loi est à queue lourde (voir section 1.1.2), et elle peut donc être utile pour caractériser les distributions de taille de transactions.
Empirique	Lorsqu'aucune des lois précédentes n'est satisfaisante on peut, en tant que modélisation, garder simplement les fréquences d'apparition de chaque valeur prise par le processus. A partir de cette information, il est possible de générer du trafic suivant la même loi empirique (voir section 3.1.1).
Covariance	
IID	Ce sont les processus les plus classiques, les variables aléatoires sont indépendantes et suivent toutes la même loi. Il n'y dans ce cas, aucun paramètre à estimer.
ARMA	Les processus ARMA (voir section 1.3.2) introduisent des dépendances à court terme, permettant de caractériser les variations du processus à petites échelles. On peut ainsi caractériser un comportement en rafale à petites échelles (*bursty behavior* en anglais).
FGN	Cette forme de covariance est utilisée pour caractériser une propriété de longue mémoire, c'est à dire un comportement en rafale à toutes les échelles (même aux plus grandes). Étant donné l'impact de cette propriété sur les performances du réseau (voir section 2.2), et le fait qu'elle a été identifiée sur puce (au niveau macro-bloc [151]), il est important de pouvoir générer du trafic ayant cette caractéristique.
FARIMA$(1,d,1)$	On peut vouloir caractériser, en plus de la longue mémoire, un comportement particulier aux petites échelles, et utiliser pour cela les processus FARIMA (voir section 2.6)

Tab. 6.2: Justification du choix des différentes forme de loi marginale et de covariance retenus pour modéliser le trafic sur puce.

saction et le type de génération du placement temporel des transactions. Nous avons défini un format de fichier, qui sera utilisé par notre générateur de trafic. Un exemple de fichier de configuration est présenté sur la figure 6.7. Dans cet exemple, deux phases ont été identifiées dans le trafic, et le concepteur a choisi différentes modélisations pour chacune d'entre elles. On peut remarquer que le format de ce fichier de configuration est très compact, ce qui permet une lecture ainsi qu'une écriture manuelle aisée.

```
phase0{ //phase numéro 0
 time: //Caractéristiques du placement temporelle des transactions
       mode=IA; //Précision délai, c'est le délai qui sera utilisé
                //pour le placement temporel des transactions
       exponential(15); //Délai modélisé par un processus IID suivant
                        //une loi exponentielle de paramètre 15
 content: //Caractéristiques du contenu des transactions
       random("ptable",exponential(10));
         //Choit d'un génération aléatoire
         //ptable est le fichier contenant la table destination/adresses
         //La taille est modélisée par un processus IID suivant
         //une loi exponentielle de paramètre 10
 duration: //Durée de la phase
       constant(10000); //Cette a une durée constante de 10000
                        //transactions
}

phase1{ // phase numéro 1
 time: // Caractéristiques du placement temporelle des transactions
       mode=TP;//précision débit, c'est le débit qui sera utilisé
               //pour placement temporel des transactions
       deterministic("fic");
              //Le débit n'est pas modélisé, une trace précédement
              //enregistrée est compressée dans le fichier fic est rejouée
 content: // Caractéristiques du contenu des transactions
       cache("pt",exponential(16));
         // Choit d'une génération de type cache (écriture non-blocantes)
         //ptable est le fichier contenant la table destination/adresses
         //La taille est modélisée par un processus IID suivant
         //une loi exponentielle de paramètre 16
 duration: // Durée de la phase
       deterministic("fichier"));
           // Les différentes durées de cette
           // phases on été stockées dans le fichier fic2
}

sequencer{ // Gestion du passage d'une phase à l'autre
  round(10); // Alternance continue entre les 2 phases, 10 fois
}
```

Fig. 6.7: Exemple de fichier de configuration de notre générateur de trafic.

6.2.5 Générateur de trafic multiphase (MPTG)

Nous avons développé un générateur de trafic sur puce générique, en SystemC et dans l'environnement SocLib (voir section 7.1). Ce générateur implémente une interface VCI (*Advanced*) et repose sur une bibliothèque C/C++ indépendante (appelée `libmptg`) qui implémente toutes les routines de génération de réalisation des différents processus stochastiques introduits dans la section précédente. Cette bibliothèque prend aussi en charge les changements de phases. L'interface entre le générateur de trafic et la bibliothèque se fait par l'intermédiaire d'un objet C++, initialisé par le fichier de configuration et qui possède une méthode `next()` renvoyant à chaque appel une nouvelle transaction $T(k) = (A(k), C(k), S(k), D(k))$. La génération des délais si la précision débit a été choisie est explicitée dans la section 6.2.5.2.

6.2.5.1 Modes de fonctionnement

Rappelons qu'un des objectifs de notre environnement de génération de trafic est de pouvoir, à partir d'une trace de référence collectée avec une autre interconnexion, générer du trafic dans une plateforme incluant une interconnexion arbitraire. Il faut pour cela s'assurer que les communications du composant que l'on va remplacer par un générateur de trafic ne changent pas fondamentalement lorsque l'on change d'interconnexion. Du point de vue du composant, cela signifie que les communications seront les mêmes, quelque soit la latence de l'interconnexion. Dans le cas général d'un processeur associé à un cache, on ne peut pas garantir que, quelque soit la latence de l'interconnexion, le contenu des transactions (les séquences $A(k), C(k)$ et $S(k)$) ne soit pas affecté par la latence du réseau. Cela est en particulier dû à la présence du tampon d'écritures postées. En effet, le fonctionnement d'un tel registre peut, dans certain cas, engendrer une modification de la taille des transactions d'écritures envoyées dans le réseau en fonction de la latence de ce dernier, en particulier dans le cas de grandes séquences d'écritures consécutives (initialisation à 0 d'une partie de la mémoire par exemple).

On fera donc l'hypothèse par la suite qu'il n'y a pas de tampon d'écritures postées. Fort de cette hypothèse, on peut affirmer que le contenu des transactions du trafic d'un processeur ne sera pas affecté par la latence du réseau. L'étude d'une émulation de ce tampon d'écritures postées est en cours de test.

Le placement temporel des transactions, lui, va être modifié par la latence du réseau, et c'est pour pourquoi nous avons utilisé le délai $D(k)$ qui est relatif à la réception de la requête, et donc indépendant de la latence du réseau. Reste cependant le problème du recouvrement des calculs et des communications. Il faut en effet pouvoir déterminer si le temps entre deux transactions $D(k)$ est un temps pendant lequel le composant attend la réponse (le composant est bloqué en attendant la réponse), ou si c'est un temps pendant lequel le composant continue à fonctionner, et donc à produire de nouvelles communications. Ceci nous a poussé à définir différents modes de fonctionnement du générateur de trafic :

- **Transactions blocantes**. Dans ce mode, quelque soit la commande, le générateur de trafic émet une rafale de type $C(k)$ à destination de l'adresse $A(k)$, de taille $S(k)$ mots. Une fois que la réponse est reçue, alors le générateur de trafic attend $D(k)$ avant d'appeler la fonction `next()` une nouvelle fois et d'envoyer une nouvelle requête. Ce mode caractérise un composant qui est bloqué (en attente) lorsqu'il fait une requête.
- **Transactions non-blocantes**. Dans ce mode, quelque soit la commande, le générateur de trafic émet une rafale de type $C(k)$ à destination de l'adresse $A(k)$, de taille $S(k)$ mots. Une fois les $S(k)$ mots envoyés, le générateur de trafic commence à compter $D(k)$. A la réception de la réponse, si $D(k)$ est déjà passé, alors la requête suivante est immédiatement envoyée, sinon on attend que $D(k)$ soit atteint. Ceci permet de modéliser un composant flot de donnée (accélérateurs matériels par exemple) qui n'est pas bloqué par les communications.
- **Lecture et écriture blocantes/non-blocantes**. On peut aussi spécifier plus précisément si les transactions en lecture et en écriture sont blocantes ou non-blocantes. Par exemple, un cache de type écriture directe n'est pas bloqué par les écritures (le processeur continue à fonctionner). En revanche, une requête de lecture bloque le processeur qui doit attendre la donnée avant de poursuivre son exécution. Le mode écriture non-blocantes, lecture blocantes est donc dont une bonne approximation du comportement d'un cache.
- **Mode flot de données**. Afin de pouvoir émuler le trafic des composants flot de données, nous avons enfin défini un mode de communication dans lequel seules les requêtes sont considérées (l'arrivée des réponses n'est pas pris en compte). Dans ce mode, le générateur

de trafic émet une requête, attend $D(k)$ cycles, et émet la requête suivante, sans se préoccuper de l'arrivée ou non de la réponse.

La définition de ces modes de fonctionnement nous permet, sans perte de généricité de l'outil, de pouvoir s'adapter aux différents types de génération de trafic sur puce.

6.2.5.2 Génération des délais en précision débit agrégé

Lorsque la précision débit est sélectionnée, on perd l'information de délai entre les transactions, et il existe plusieurs moyens de répartir les transactions dans la fenêtre de taille Δ :

- **Au plus tôt**. Des requêtes sont envoyées les unes à la suite des autres jusqu'à ce que la somme des tailles des transactions émises soit égale au débit visé pour la fenêtre en cours ($W_\Delta(i)$).
- **Délai constant**. Les requêtes sont réparties dans la fenêtre de manière à ce que le délai entres elles soit constant (pour la fenêtre en cours). Nous avons opté pour cette solution qui produit un trafic plus réaliste.

6.2.5.3 Développements

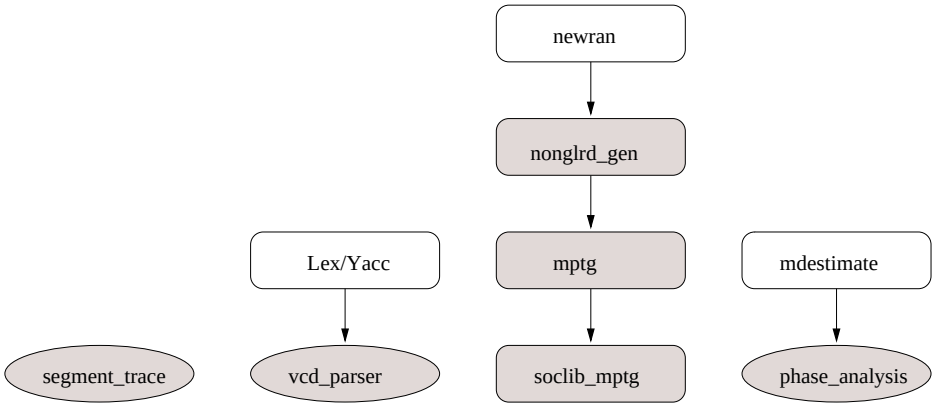


Fig. 6.8: Les différentes bibliothèques (carrés) et programmes (ellipses) de notre environnement de génération de trafic sur puce.

Afin d'offrir au lecteur une vision claire de l'organisation logiciel de l'environnement de génération de trafic sur puce que nous avons développé, la figure 6.8 montre les différentes bibliothèques et programmes (les partie grisées ont été développées par nos soins). Le développement est modulaire pour que les parties puissent être individuellement utilisées dans d'autres contexte. Toutes ces librairies ont été écrites en langage C++ et en voici une description rapide :

- `newran`. C'est une bibliothèque de génération de nombres aléatoires suivant différentes lois écrite par Robert Davis [36]. Nous l'avons utilisée pour générer des réalisations de processus IID, et elle a aussi servie de base à la bibliothèque `nonglrd_gen`.
- `nonglrd_gen`. Cette librairie permet de générer des réalisations de processus à longue mémoire suivant différentes lois marginales. Elle comporte 1300 lignes de code environ. Une documentation est disponible en ligne :
  http://perso.ens-lyon.fr/antoine.scherrer/nonglrd_gen_doc/.

- `mptg`. Cette librairie implémente les différents modes de génération introduit dans la section 6.2.4 et inclus un analyseur syntaxique pour lire un fichier de configuration MPTG (voir figure 6.7). Elle comporte 2000 lignes de code environ. Une documentation est disponible en ligne :
  http://perso.ens-lyon.fr/antoine.scherrer/mptg_doc/
- `soclib_mptg`. Il s'agit du composant soclib de génération de trafic que nous avons développé. Il est écrit en SystemC et implémente les différents modes de génération discutées dans la section 6.2.5.1. Il fait 600 lignes de code environ.
- `vcd_parseur`. Il s'agit du programme permettant de lire un fichier au format VCD et d'en extraire la séquence de transaction $T(k)$. Il est basé sur les outils FLEX et BISON. Il comporte environ 1500 lignes de code. Une documentation est disponible en ligne :
  http://perso.ens-lyon.fr/antoine.scherrer/vcd_parser_doc/
- `segment_trace`. C'est le programme qui effectue la segmentation de phase explicitée dans la section 6.1.3. Il comporte environ 1600 lignes de code. Une documentation est disponible en ligne :
  http://perso.ens-lyon.fr/antoine.scherrer/seg_doc/
- `phase_analysis`. C'est le programme qui effectue la modélisation d'une série (estimation des paramètres du modèle choisi) comme expliqué dans la section 6.2.4.1. Il comporte environ 1300 lignes de code. La documentation de ce programme est en cours d'écriture.

Ces développements ont été effectués sous licence GPL, ils constituent une contribution importante de la thèse.

6.2.6 Exploration architecturale et évaluation de performance

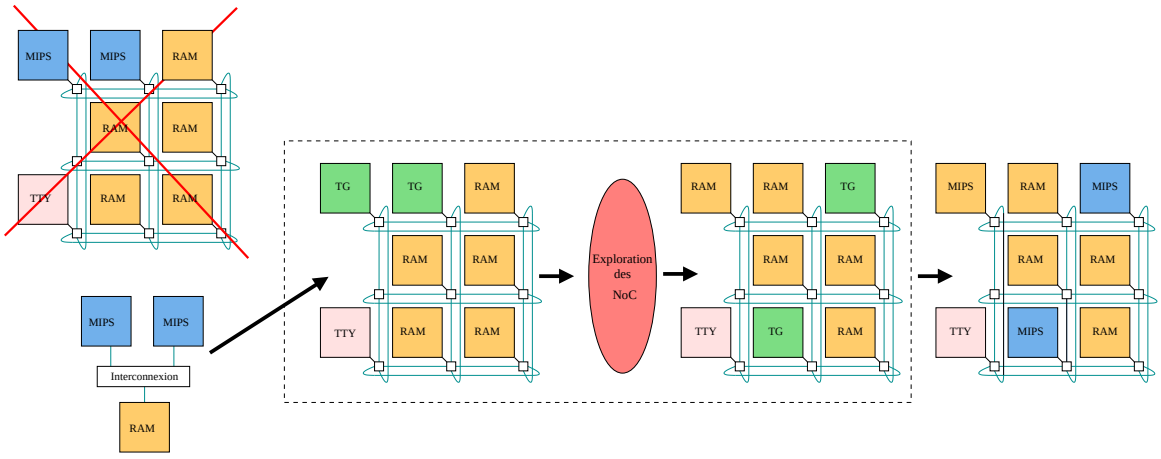


Fig. 6.9: Flot d'exploration architecturale. Exemple de changement de placement des composants afin d'améliorer les performances.

Nous avons maintenant défini le flot permettant d'aller d'une simulation de référence avec des composants réels, vers une simulation ou les composants sont remplacés par des générateurs de trafic. Avec ces générateurs de trafic, le concepteur pouvoir procéder à l'exploration architecturale du réseau d'interconnexion. Cette évaluation sera guidée par les évaluations de performances effectuées avec des générateurs de trafic correctement configurés. On peut citer, parmi les choix architecturaux que le concepteur du NoC doit faire :

- **Configuration du réseau**. Tous les paramètres décrits dans la section 5.2.2 peuvent être ici explorés en utilisant des générateurs de trafic. Cela va des choix à large échelle (topologie, stratégie de routage, etc.), au choix fins comme le dimensionnement des *buffers* par exemple.
- **Placement des composants**. Dans un contexte de réseau sur puce, le placement des différents composants dans la topologie du réseau est à définir. Comme cela a déjà été dit plus haut, les communications sont très locales dans les SoC, c'est à dire que l'on gagne beaucoup à faire ce placement de manière réfléchi. On peut, pour guider ce choix, calculer des matrices de trafic représentant la quantité de trafic échangée entre chaque paire de composants [30]. On peut aussi utiliser notre environnement comme cela est illustré sur la figure 6.9. Cette figure illustre le fait que la simulation de référence est effectuée sans interconnexion pour être plus rapide. En suivant notre flot (voir figure 6.4), on dérive des générateurs de trafics en remplacement des composants initiateurs. On peut alors effectuer différentes simulation et en déduire le placement optimal des composants. Enfin pour effectuer une validation on replace les composants dans la plateforme.

6.3 Points clés de notre générateur de trafic

Nous résumons dans cette section les caractéristiques importantes de notre environnement de génération de trafic.

- **Rejeu d'une trace enregistrée**. C'est la fonctionnalité de base d'un générateur de trafic. Le point important est qu'une trace enregistrée sur une interconnexion quelconque (ou même sans interconnexion) peut être utilisée pour rejouer le trafic dans une autre plateforme comme cela a été discuté dans la section 6.2.4, et sera mis en évidence par des résultats expérimentaux dans la section 8.2. Ceci permet aussi à des concepteurs d'IP de diffuser un générateur de trafic à la place du composant, évitant ainsi des problèmes de propriété intellectuelle.
- **Génération de trafic stochastique**. L'utilisation de processus stochastique pour générer du trafic permet de générer sur le réseau une charge aléatoire ayant des caractéristiques statistiques données. Nous avons pour cela inclus une large gamme de modèle de processus stochastique (voir section 6.2.4.1) afin de pouvoir s'adapter à de nombreuses situations. Les expérimentations illustrant cette fonctionnalité sont présentées dans la section 8.4.
- **Génération multiphase**. Notre générateur de trafic peut générer du trafic défini par plusieurs phases correspondant à des modèles différents comme cela a été discuté dans la section 6.2.3. Les résultats de segmentation automatique sont présentés dans la section 8.3.
- **Prise en compte de la longue mémoire**. Notre environnement intègre des modèles de processus stochastiques à longue mémoire, et peut donc simuler cette propriété sur un réseau. Les résultats sur ce point précis sont discutés dans la section 8.5.
- **Procédure de d'estimation des paramètres**. Notre environnement inclut une procédure pour estimer les meilleurs paramètres d'un modèle de processus stochastique donné, étant donnée un morceau d'une trace de référence. Cela est fait automatiquement, mais la sélection du modèle est pour l'instant laissé au concepteur.
- **Modélisation du trafic**. Nous avons fait le choix de laisser la possibilité de modéliser, de manière indépendante, le placement temporel des transactions, et leur contenu (destination, commande, taille).

- **Mode de génération**. Afin de pouvoir s'adapter aux modes de communication des différents composants d'un SoC, nous avons défini différents modes de communications (voir section 6.2.4).

En conclusion, nous avons défini un environnement de génération de trafic flexible, incluant de nombreuses fonctionnalités qui ont été présentées dans différents travaux récents ou introduites par nos soins. Nous avons aussi défini une méthodologie afin de pouvoir, à partir d'une trace de référence, générer semi-automatiquement la configuration d'un générateur de trafic qui pourra être utilisée ensuite dans une plateforme quelconque.

La chapitre suivant présente l'environnement de simulation que nous avons utilisé pour les expérimentations qui serons détaillées dans le chapitre 8.

Chapitre 7
Environnement de simulation

Ce chapitre présente l'environnement de simulation que nous avons utilisé pour effectuer nos expérimentations. La section 7.1 décrit l'environnement de simulation SOCLIB, puis la section 7.2 montre le flot de simulation avec cet environnement. Enfin la section 7.3 décrit les applications que nous avons utilisées.

7.1 SocLib

Cette section présente l'environnement de simulation de SoC, SOCLIB. La section 7.1.1 décrit le fonctionnement de SOCLIB ainsi que la modélisation des composants et la section 7.1.2 liste les composants disponibles dans cet environnement.

7.1.1 Fonctionnement

SOCLIB [114] est un environnement de simulation de système sur puce libre (*open-source*), basé sur le langage et le moteur de simulation SYSTEMC. Il a été initié par l'équipe ASIM du laboratoire LIP6, et aujourd'hui de nombreuses autres équipes françaises (LIP, LIRMM, TIMA, ENST, etc.) l'utilisent et y contribuent.

SOCLIB est une bibliothèque de composants pouvant être simulés au cycle près et au *bit* près (CABA pour *Cycle Acurate, Bit ACcurate*). Il est prévu de pouvoir aussi effectuer des simulations de niveau TLM, mais cela est encore en cours de développement. Les composants sont synchrones, c'est à dire qu'il ont tous un signal d'horloge qui cadence leur fonctionnement.

Le composant est modélisé par un ou plusieurs automates d'état finis (FSM pour *Finite State Machine*) de Moore ou de Mealy [28]. Le composant est donc modélisé par un ensemble de registres et de signaux d'entrées/sorties qui sont manipulés par trois fonctions décrites ci-après et illustrées sur la figure 7.1.

- **Transition**. Cette fonction calcule, à partir de l'état courant et des valeurs courantes des signaux d'entrées, l'état suivant des différents automates. Cette fonction est exécutée à chaque front montant du signal d'horloge du composant.
- **Moore**. Cette fonction gère les automates dits de Moore, pour lesquels les sorties (valeurs de signaux de sortie) sont déterminées uniquement en fonction de la valeur de l'état de l'automate. Cette fonction est exécutée sur le front descendant de l'horloge, donc après la fonction de transition.
- **Mealy**. Cette fonction gère les automates dits de Mealy, pour lesquels les sorties sont déterminées en fonction de l'état de l'automate *et* de la valeur des entrées. Cet automate est sensible aux modifications des signaux d'entrées, c'est pourquoi il doit être exécuté plusieurs fois, jusqu'à ce que ses signaux d'entrées/sorties soient stables.

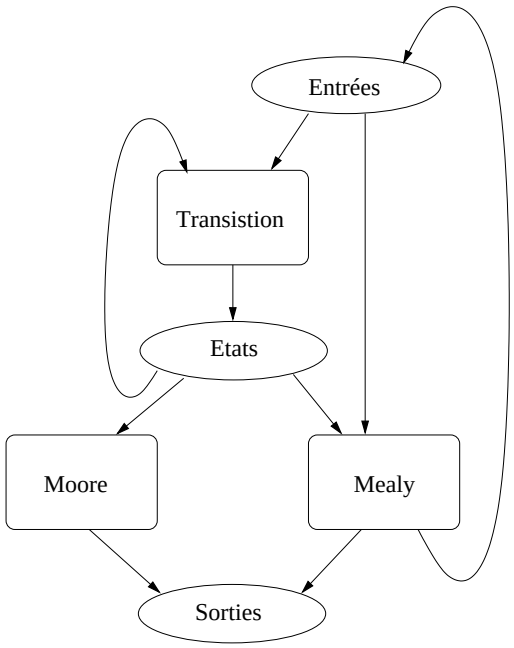


Fig. 7.1: Modélisation des composants dans l'environnement SocLib.

Ce cycle se répète ensuite et l'automate évolue au cours du temps, au rythme de l'horloge du composant. Il est à noter que les composants peuvent disposer de signaux d'horloge différents, et on peut donc simuler des systèmes localement synchrones et globalement asynchrones (GALS, voir section 5.1.1).

Afin de pouvoir simuler des SoC complets, il faut pouvoir interconnecter les différents composants, et SocLib a pour cela fait le choix d'utiliser l'interface standard VCI (voir section 5.3.2). Tous les composants SocLib disposent donc d'une interface VCI (*Basic* ou *Advanced*) et la gestion du protocole de communication est réalisée par un FSM de Moore.

7.1.2 Composants disponibles

Cette section décrit les différents composants disponibles dans SocLib [114]. Les composants suivis du symbole † ont été utilisés dans nos expérimentations et ceux suivis du symbole ‡ ont fait l'objet de développements par nos soins.

Processeurs

- **MIPS R3000** ††. Les MIPS sont de petits processeurs à petit jeu d'instruction (RISC pour *Reduced Instruction Set Computer*), très utilisés dans le domaine des systèmes embarqués. Le MIPS R3000 est relativement simple, il possède un *pipeline* à deux étages, et la version utilisée dans SocLib ne dispose pas d'unité pour effectuer des calculs sur les flottants. C'est ce processeur que nous avons utilisé dans toutes nos simulations.
- **Cache** †. Le MIPS, afin d'être intégré dans SocLib, doit être couplé à un cache. Le cache dans SocLib est à correspondance direct, et à écriture directe. La géométrie du cache (nombre de mots par ligne et nombre de lignes) est paramétrable. On peut aussi spécifier des plages d'adresses non-cachées (voir section 5.3.3.2). Ce cache dispose d'une interface VCI *Advanced*.

- **Autres processeurs**. Un ARM et un PowerPC sont en cours de développement.

Générateur de trafic

- **Générateur de trafic multiphase** ‡†. Notre MPTG a naturellement été intégré dans SoCLib. Il est compatible avec l'interface standard VCI, mais il a été développé de manière à s'adapter facilement à d'autres interfaces standards. C'est un composant très simple (600 lignes de codes environ), car comme cela a déjà été mentionné dans la section 6.2.5, la génération des réalisations de processus stochastique a été développée dans une bibliothèque indépendante. Les caractéristiques importantes de ce composant sont résumées dans la section 6.3.

Accélérateurs matériels

- **Accès direct à la mémoire** ‡. Un DMA[1] a été développé dans SocLib pour que les accélérateurs matériels puissent accéder à la mémoire sans passer par le processeur.
- **Interface** ‡. Afin de simplifier le développement d'accélérateurs matériels, nous avons défini une interface générique [FRS04] paramétrable. Cette interface utilise le DMA [FRS04].
- **Filtrage FIR** ‡. Un composant effectuant un filtrage à réponse impulsionnelle fini (FIR pour *Finite Impulse Response*) a été développé pour évaluer cette interface.
- **Transformée en ondelette** ‡. Afin d'accélérer le décodeur d'image JPEG2000 (voir section 7.3), nous avons développé un accélérateur matériel qui effectue rapidement l'étape de transformée en ondelette discrète inverse. Ce travail est en cours de finalisation et fera prochainement l'objet d'une publication.

Mémorisation

- **Mémoire simple** †. Une mémoire simple est incluse dans SocLib, le temps d'accès aux données n'est pas précisément simulé (la mémoire répond en un cycle).
- **Mémoire de sémaphore** †. Cette mémoire, en plus de stocker des informations, simule le fonctionnement de sémaphores matériels utilisés pour gérer les exclusions mutuelles entre les threads par exemple.

Interconnexions

- **Crossbar**. Ce composant simule une interconnexion générique (double crossbar, un pour les requêtes et un autre séparé pour les réponses). La latence de cette interconnexion est fixe (il n'y a jamais de contention). Elle peut être utilisée pour effectuer des tests, car elle ne représente pas une interconnexion réalisable pour un SoC (beaucoup trop coûteuse en terme de surface et de consommation).
- **Interconnexion générique** †. Ce composant est similaire au crossbar, mais la latence peut être paramétrée (simulé à l'aide d'une file d'attente).
- **SPIN** †. Il s'agit du réseau sur puce développé par le LIP6 (voir section 5.2.3.2), disposant d'une topologie en arbre élargi, d'un routage statique (avec séparation des chemins pour les requêtes et les réponses) et d'une mémorisation de type trou de ver (*wormhole*).
- **DSPIN** †. Il s'agit de l'évolution de SPIN, qui est une grille 2D avec un protocole de routage

[1] *Direct Memory Access*

statique de type XY et une mémorisation trou de ver (voir section 5.2.3.2).

Périphériques

- **Interface série** †. Une interface série permet d'afficher des messages (caractère par caractère) dans un terminal et/ou dans un fichier.
- **Affichage vidéo** †‡. Ce composant permet d'afficher une image ligne par ligne. L'afficheur est simulé avec précision car les temps de retour à la ligne et de retour en haut de l'affichage sont simulés (et paramétrables).
- **Timer**. Un *timer* est un composant envoyant un signal d'interruption périodiquement tout les n cycles.

Autres

- **Table de segment** †. Cette table contient la définition de l'organisation mémoire, c'est à dire une liste de segments (morceaux de l'espace d'adressage), avec pour chacun d'entre eux des informations (mémoire qui le stocke, type de segment, taille, etc.). Cet élément est partagé par la plupart des composants.

Plateformes

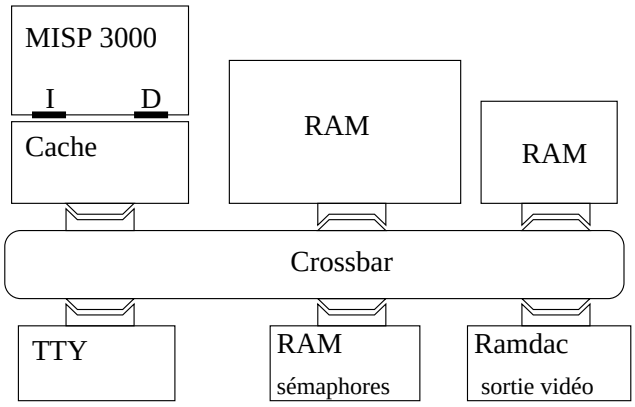


Fig. 7.2: Exemple de plateforme pouvant être simulée dans l'environnement SocLib.

On peut assembler ces composants pour former un SoC complet, pour cela il faut connecter leurs signaux d'entrées sorties entres eux, ce qui est réalisé dans un fichier SystemC qui instancie les différents composants et les interconnecte. Ce fichier inclut aussi la boucle de simulation qui consiste à générer l'évolution du ou des signaux d'horloge qui vont provoquer l'exécution des différentes fonctions de chaque composant décrites dans la section précédente. Un exemple de plateforme SocLib est représentée sur la figure 7.2.

7.1.3 Générateur de plateforme

Afin d'accélérer le temps de mise en place d'une plateforme, nous avons développé un outil que nous avons appelé `socgen`, permettant, à partir d'une description simple et compacte des composants que l'on souhaite intégrer, de générer le fichier SystemC associé ainsi que les différents scripts (*Makefile*, édition des liens, etc.) permettant de compiler le simulateur et l'ap-

```
proc_number 2 // 2 processeurs
platform_name p1 // Nom de la plateforme
cache_params custom // Paramètres des caches
// modeI/D : activation du cache d'instructions / données
  // 0 = actif, 1 = inactif
// lineI/D : Nombre de ligne de cache
// wordI/D : Nombre de mot dans une ligne
// Nom modeI modeD lineI wordI lineD wordD
  cache0 0 0 32 8 32 8
  cache1 0 0 32 8 32 8
trace yes // Active la sortie VCD
gen_tg no // Génère une plateforme avec des générateurs de traffic
          // à la place des processeurs
mem_mapping custom // Mapping mémoire
// On affecte chaque segment (codeN, inputN, outputN)
// à une mémoire (ramN)
  code0 ram0
  output0 ram1
  input0 ram2
  code1 ram3
  output1 ram3
  input1 ram4
interconnect dspin // Interconnexion
// Paramètres de l'interconnexion
  req_fifo_size 8 // FIFO requêtes
  rsp_fifo_size 8 // FIFO réponses
// Placement des composants
// Composant X Y
  mips0 0 2
  mips1 1 2
  ram0  2 0
  ram1  1 0
  ram2  2 1
  ram3  1 1
  ram4  2 2
  tty   0 0
```

Fig. 7.3: Fichier de description de plateforme SocLib et la plateforme associée.

plication (voir section 7.2). Nous avons pour cela défini un format simple de description de plateforme. Un exemple de fichier est décrit sur la figure 7.3 et de la plateforme associée est représentée sur la figure 7.4. La description inclus la majorité des paramètres des différents composants ainsi que le placement (si c'est nécessaire) des composants dans le réseau. Il est ainsi aisé d'explorer ce placement, qui, sans cet outil, est un travail très long puisqu'il faut connecter chaque commutateur à la bonne ressource. L'organisation mémoire est dérivée d'une configuration standard statique (par exemple le segment `code1` sera toujours affecté à l'adresse `0x10400000` et aura une taille de `0x10000` octets[2]). Nous n'avons pas ressentit le besoin, pour nos simulations, de laisser la possibilité de spécifier ces affectations dans le fichier de configuration. Une telle extension est néanmoins envisageable. L'outil `socgen` est écrit en PERL.

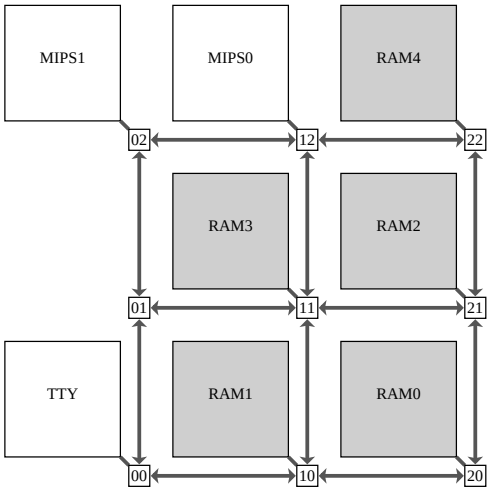


Fig. 7.4: Plateforme SocLib correspondant au fichier de configuration de la figure 7.3.

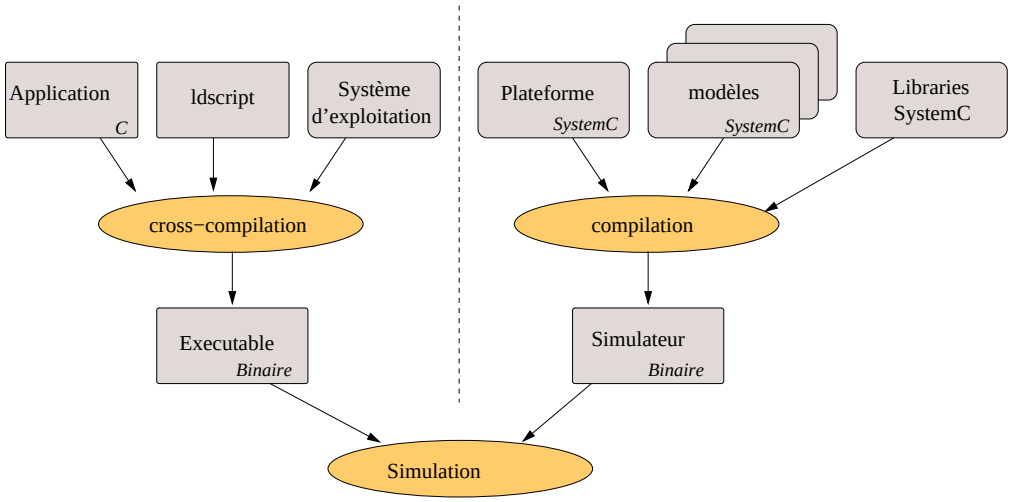


Fig. 7.5: Flot de simulation d'une plateforme SocLib.

7.2 Flot de simulation

Effectuer la simulation d'un SoC dans SocLib implique différentes étapes représentées sur la figure 7.5 et décrites ci-après.

- **Compilation de l'application** (partie gauche de la figure 7.5). Le code de l'application que l'on souhaite exécuter sur le ou les processeurs de la plateforme est compilé à l'aide de GCC (GNU *C Compiler*). L'application peut nécessiter un système d'exploitation (OS), dont le rôle est de gérer les périphériques, les accès à la mémoire (gestion des sémaphores par exemple) et les communications entre les différents threads. SocLib inclut un petit système d'exploitation multithread et multiprocesseur : MUTEK [126]. L'ensemble (application et OS) est *cross-compilé*, c'est à dire que le code binaire produit est exécutable sur un processeur donné comme un MIPS par exemple. Il est aussi nécessaire de décrire l'organisation de la mémoire dans un script utilisé lors de l'édition des liens (*ldscript* est le format de script utilisé par GCC). On obtient alors un exécutable contenant l'application

[2] La notation 0xN indique que le nombre N est écrit en base 16 (hexadécimal).

et le système d'exploitation, pour une organisation mémoire décrite dans le *ldscript*.

- **Compilation du simulateur** (partie droite de la figure 7.5). SOCLIB est basé sur le moteur de simulation SYSTEMC, qui est lui même basé sur le langage C++. Pour obtenir le simulateur (exécutable sur la station de travail), il faut compiler ensemble la description de la plateforme qui est associée aux modèles des différents composants utilisés qu'elle intègre. Cette compilation est aussi réalisée à l'aide de GCC.
- **Simulation**. La simulation peut maintenant avoir lieu, le simulateur va instancier les différents composants et entrer dans la boucle de simulation. Lors de la simulation, on peut enregistrer les variations de signaux dans un fichier au format VCD (*Value Change Dump*), ce qui va nous permettre d'obtenir la trace de référence comme cela a été expliqué dans la section 6.2.2.

Il est important de noter que ce flot est légèrement modifié par l'utilisation du générateur de plateformes (`socgen`), voir section 7.1.3. En effet, dans ce cas on n'écrit pas manuellement le fichier de description de la plateforme, ni le script d'édition des liens de l'application, tout ceci est généré par `socgen` comme cela est illustré sur la figure 7.6.

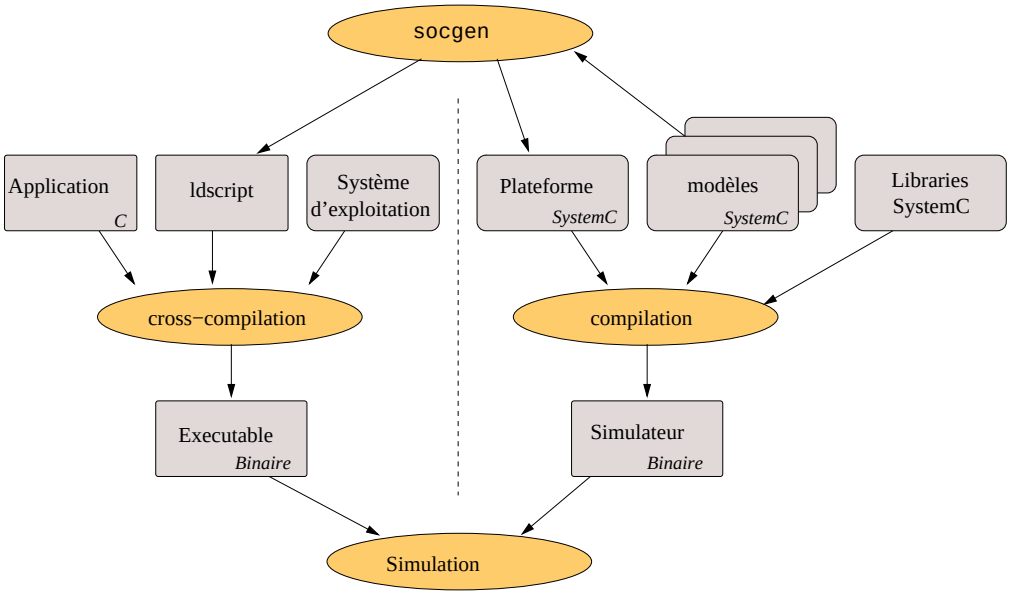


Fig. 7.6: Flot de simulation d'une plateforme SocLib avec le générateur de plateformes.

7.3 Applications

Cette section décrit les différentes applications que nous avons utilisées pour valider notre flot de génération de trafic sur puce. Nous avons cherché à utiliser des programmes réellement utilisés dans les SOC, mais n'ayant pas accès aux applications d'industriels, nous avons utiliser des implémentations *open-source* disponible sur le WEB.

Afin de pouvoir exécuter ces applications sur une plateforme SOCLIB, quelques modifications décrites ci-après ont du être effectuées.

- **Compatibilité avec MUTEK**. Les applications que nous avons utilisées sont originellement écrites pour fonctionner avec un système d'exploitation de type LINUX alors que

MUTEK, le système d'exploitation que nous avons utilisé, est beaucoup plus simple. Il a donc fallu simplifier quelques parties du code comme la gestion des erreurs avec l'utilisation de la bibliothèque `setjmp` qui n'est pas implémentée dans MUTEK par exemple.

- **Entrées/Sortie**. Les applications originales utilisent des fichiers comme entrées et sortie. MUTEK n'implémente pas de système de fichier, il a donc fallu changer les accès aux fichiers par des accès mémoire classiques. Les fichiers d'entrées sont chargés dans un segment mémoire au démarrage de l'application (ce chargement ne compte pas dans le temps de simulation). A la fin de la simulation, il est possible de récupérer un segment mémoire dans un fichier afin de vérifier que les résultats de l'exécution de l'application sont corrects.
- **Types de données**. Le processeur MIPS R3000 modélisé dans SOCLIB ne dispose pas d'unité pour effectuer des calculs sur des flottants, il est donc nécessaire de convertir les calculs en virgule fixe. Cette étape à été réalisée de manière *ad hoc*, car les applications que nous avons considérées sont simples. Le problème de la conversion virgule fixe/ virgule flottante est complexe et de nombreuses recherches sont effectuées sur ce sujet (voir [105] et les références incluses).

Voici la liste des applications que nous utilisées, nous nous sommes concentrés sur des applications multimédia qui engendrent beaucoup de communications à cause des traitements intensifs que ces algorithmes demandent.

- **JPEG** (*Joint Photographic Expert Group*). Il s'agit de l'implémentation classique utilisée sous LINUX [56] de l'algorithme de décompression d'image le plus utilisé à l'heure actuelle. Il comporte les étapes suivantes : décodeur de Huffman, déquantification, transformée cosinus discrète inverse, remise des coefficients dans l'ordre, affichage.
- **MJPEG** (*Muli*-JPEG). C'est un format vidéo dans lequel toutes les images sont simplement compressées avec l'algorithme JPEG. Il s'agit cependant d'une implémentation multitread développé par le LIP6 [130], dans laquelle six threads communiquent par des canaux de communications abstraits définis dans le système d'exploitation. Chaque étape du décodage est effectué dans un thread différent, et peut donc être exécuté par différents processeurs en parallèle. Nous utiliserons cette application plus comme une version multitreads de JPEG, que comme la décompression d'un flux vidéo car nous considérerons dans nos expérimentations le décodage d'une seule image.
- **JPEG2000**. C'est le standard de compression d'image qui devrait à terme remplacer JPEG. La transformée cosinus a été remplacée par une analyse multirésolution (transformée en ondelettes discrète) et le codage des données est beaucoup plus complexe. Nous avons utilisé une implémentation libre et légère de cet algorithme [72].
- **MP3**. Le format MP3 est le format de compression audio le plus répandu. Nous avons, comme pour les applications précédentes, considéré le décodage à partir d'une implémentation *open-source* (`libmad` [91]). Pour que la musique puisse commencer à être écoutée avant que tout le fichier ne soit décodé, les données sont traitées par trames.

Avec ces quatre applications, nous avons une bonne vue sur les classes d'applications multimédia pouvant être utilisés dans les SOC.

Chapitre 8
Résultats expérimentaux

Ce chapitre rassemble les résultats expérimentaux que nous avons obtenu avec l'environnement de simulation SOCLIB afin de montrer la pertinence, la validité et la précision de notre méthodologie de génération de trafic présentée dans le chapitre précédent.

Ce chapitre est organisé comme suit : les générateurs de trafic sont en général développés pour faire des simulations plus rapides, ce point est exploré dans la section 8.1. La validité de la génération d'un trafic précédemment enregistré est ensuite illustrée dans la section 8.2. Les résultats concernant la segmentation automatique en phases sont détaillés et analysés dans la section 8.3 et les résultats concernant la modélisation du trafic par processus stochastique sont présentés dans les sections 8.4. Enfin, l'aspect longue mémoire est exploré dans la secction 8.5 et un résumé des contributions est proposé dans la section 8.6.

8.1 Temps de simulation

Cette section présente des résultats concernant le gain en temps de simulation induit par le remplacement des IP par des générateurs de trafic. C'est, à l'origine, la principale cause de l'utilisation de générateurs de trafic, mais nous allons montrer que ce gain n'est, en pratique, pas très important, et que le réel intérêt d'un outil de génération de trafic ne peut se résumer à cela. La section 8.1.1 présente comment le temps d'une simulation est réparti, la section 8.1.2 décrit les simulations que nous avons effectuées, et la section 8.1.3 présente et discute les résultats.

8.1.1 Temps de simulation avec SOCLIB

Le temps d'une simulation SOCLIB peut être décomposé en différentes parties décrites ci-dessous.

• **Simulation SYSTEMC**. Cela correspond à l'exécution du moteur de simulation SYSTEMC (simulation à événements discrets). On peut considérer que cette proportion du temps de simulation est constante avec des générateurs de trafic ou avec des processeurs.

• **Simulation des composants**. C'est le temps qui est passé à exécuter les fonctions des différents composants (transition, automates de Moore, automates de Mealy, voir section 7.1.1). Ce temps dépend surtout de la finesse de la modélisation (il dépend aussi de l'activité du composant). Un composant modélisé de manière grossière sera simulé plus rapidement que si son comportement interne est simulé en détails. Le processeur MIPS r3000 que nous avons utilisé dans ces simulations fonctionne comme un simulateur d'instruction, le temps passé à le simuler n'est donc pas très important. On pourrait s'attendre à des facteurs d'accélération plus importants si ce processeur était très finement modélisé (chemin de données, pipeline) ou si on avait utilisé un processeur plus complexe. Le temps gagné à le remplacer par un générateur de trafic

aurait alors été supérieur. L'interconnexion est aussi un composant, et le temps passé à sa simulation dépend du réseau lui-même, de la charge que l'on fait supporter au réseau ainsi que de la finesse de modélisation des éléments de celui-ci. L'activité (la charge) du réseau est fonction du nombre de composants raccordés et de leur communications, et on peut logiquement s'attendre à ce que la simulation du réseau prenne une part importante de la simulation, surtout dans le cas d'un NoC à commutation de paquet. Les résultats concernant le temps de simulation avec générateurs de trafic et avec processeurs sont détaillés dans la section suivante.

• **Écriture du fichier VCD**. C'est le temps passé à enregistrer les variations de certains signaux dans un fichier au format VCD. Ce temps est important car les tailles de ces fichiers peuvent atteindre plusieurs giga-octets. Cet enregistrement est nécessaire pour collecter la trace de référence. Pour les simulations utilisant le générateur de trafic, on peut l'activer pour effectuer une comparaison entre la simulation avec générateurs de trafic et celle de référence. Une fois que la génération de trafic est validée, l'enregistrement VCD n'est plus nécessaire et permet de diminuer sensiblement le temps de simulation.

8.1.2 Simulations effectuées

Afin d'évaluer le gain en temps de simulation induit par le remplacement des processeurs par des générateurs de trafic, nous avons effectué différentes simulations sur une machine *Bi-Xeon* disposant de 2Go de mémoire vive. La durée des simulations a été fixée à 10 millions de cycles. Les différentes simulations effectuées sont décrites ci-après.

• **Plateformes de simulation**. Nous avons utilisé le réseau sur puce DSPIN dans ces simulations, la taille de la grille est reportée dans la ligne "Réseau" du tableau 8.1. La taille "0x0" correspond à la simulation initiale, effectuée sans interconnexion. Les figures 8.1(a) et 8.1(b) montrent respectivement l'exemple d'une plateforme "3x3" et "0x0".

• **Simulations avec des processeurs** (lignes MIPS). Nous avons effectué deux simulations : une enregistrant les variations de signaux (avec VCD) et l'autre sans enregistrer les variations de signaux (sans VCD). Le MIPS (associé à un cache d'instructions et de données dont la géométrie à été fixée à 32 lignes de 8 mots) exécute l'application de décodage audio MP3 (voir section 7.3).

• **Simulations avec des générateurs de trafic** (lignes MPTG). Nous avons effectué des simulations avec différentes configurations de notre générateur de trafic, afin de pouvoir évaluer leurs gains en temps de simulation. Dans le tableau 8.1, "sto." indique que la génération est basée sur des processus stochastiques IID, "det." indique un rejeu de la trace de la simulation avec les processeurs, et "lrd." indique une génération impliquant des processus à longue mémoire. Lorsque qu'il y a 10 phases, la durée des phases est réduite afin d'évaluer l'impact de la gestion du multiphase sur le temps de simulation. En effet, la simulation comporte dans ce cas un nombre important de changement de phases. Afin de pouvoir faire une comparaison équitable, les générateurs de trafic ont été configurés pour produire une charge moyenne sur le réseau identique à celle induite avec les processeurs. En effet, le temps de simulation est, comme cela a été expliqué dans la section précédente, fonction de l'activité des différents composants, il convient donc que cette activité soit similaire pour effectuer une comparaison ayant du sens.

• **Simulations de Mahadevan *et al.*** (ligne Mahadevan). Nous avons ici reporté les résultats d'accélération des travaux de Mahadevan *et al* [98] (voir section 5.4.1), qui remplace un processeur (un ARM) par un processeur beaucoup plus simple émulant le comportement en terme de communications du processeur. Ces simulations sont effectuées sur un bus AMBA et les facteurs d'accélération reportés sont donnés à titre indicatif puisqu'ils correspondent à la simula-

tion d'applications différentes (voir [98]).

8.1.3 Résultats et discussion

Nombre de processeurs	1		2		3		4	
Réseau	0x0		2x2		3x3		4x4	
	Tps.	**Acc.**	Tps.	**Acc.**	Tps.	**Acc.**	Tps.	**Acc.**
Mips sans VCD	36.1	**1**	249.5	**1**	477.5	**1**	1261.3	**1**
Mips avec VCD	59.4	**0.61**	279.9	**0.89**	559.8	**0.85**	1263.8	**0.95**
MPTG det.	15.9	**2.27**	177.2	**1.41**	344.5	**1.39**	804.0	**1.57**
MPTG 1 phase sto.	19.9	**1.82**	177.4	**1.41**	337.1	**1.42**	790.8	**1.59**
MPTG 10 phase sto.	19.9	**1.81**	177.2	**1.41**	337.6	**1.41**	790.0	**1.60**
MPTG 1 phase lrd	28.8	**1.25**	180.2	**1.39**	364.1	**1.31**	806.5	**1.56**
MPTG 10 phase lrd	29.1	**1.24**	184.0	**1.36**	341.5	**1.40**	826.4	**1.53**
Mahadevan [98]	-	**2.15**	-	**2.64**	-	**2.60**	-	**3.05**

Tab. 8.1: Temps de simulation en secondes (Tps) et facteur d'accélération (Acc.) pour différentes plateformes et différents mode de génération de trafic.

Les temps de simulation ainsi que les facteurs d'accélération sont reportés dans le tableau 8.1. Le facteur d'accélération (Acc. dans le tableau 8.1) correspond au rapport entre le temps de simulation et le temps de la simulation de référence (Mips sans VCD). Pour la ligne "Mips avec VCD" il est normal qu'il soit inférieur à 1 car l'enregistrement VCD prend du temps. Les résultats du tableau 8.1 montrent que le facteur d'accélération n'est jamais supérieure à 2.27, atteint pour la plateforme sans interconnexion (colonne "0x0"). Ceci est logique puisque dans ce cas aucun temps n'est passé à simuler l'interconnexion, le gain pour cette plateforme est donc forcément plus élevé que ceux pour des plateformes incluant des NoC. On peut noter aussi que le facteur d'accélération augmente avec le nombre de processeurs, malgré le fait que la taille de l'interconnexion (et donc son temps de simulation) augmente aussi. L'accélération est donc plus intéressante pour la simulation de grandes plateformes. Ceci est d'autant plus vrai que ces simulations sont plus longues et donc que le gain en temps réel est important.

L'impact de l'enregistrement des variations de signaux décroît avec la taille de la plateforme, ce qui est logique puisque le temps d'enregistrement reste constant alors que le temps de simulation augmente fortement. Cet impact est même négligeable pour une plateforme à quatre processeurs et un réseau "4x4".

La génération de réalisations de processus stochastiques (lignes "Sto.") n'est pas très coûteuse en terme de temps de simulation puisque le facteur d'accélération est comparable à celui d'un rejeu pour lequel il n'y a pas de génération de nombres aléatoires. La génération de réalisations de processus à longue mémoire (discutée dans les section 2.8 et 3) prend plus de temps, le facteur d'accélération est donc dans ce cas légèrement inférieur. La gestion de l'aspect multiphase ne prend pas un temps significatif puisque les facteurs d'accélération avec une ou dix phases sont toujours équivalents.

En conclusion, et c'est un point très important selon nous, on peut affirmer que dans notre environnement de simulation, l'utilisation de générateurs de trafic à la place des processeurs ne permet pas d'obtenir une accélération importante des simulations, le reste des composants continuant à être simulés par SoCLib comme avant. Nos résultats sont du même ordre de gran-

deur que ceux de Mahadevan *et al* [98] (voir section 5.4.1), et cela nous incite à penser que l'avantage de l'utilisation de générateurs de trafic à la place de processeurs n'est pas situé dans le facteur d'accélération qui est loin d'être déterminant. Il faudrait pour cela que le temps de simulation soit un ordre de grandeur en dessous, ce qui signifie des facteurs d'accélération de l'ordre de la dizaine. Or, comme cela a déjà été mentionné plus haut, le temps de simulation de l'interconnexion limite l'accélération que l'on peut espérer. Nos simulations et celles de [98] (même si leur conclusion est opposée) montre bien que de tels facteurs ne sont en pratique pas atteints. Il est à noter que les résultats de [98] sont obtenus avec un bus AMBA, ce qui explique que les facteurs d'accélération soient plus élevés puisque quelque soit le nombre de processeur, l'interconnexion reste la même.

L'avantage d'un générateur de trafic réside dans la flexibilité de l'outil qu'il représente pour le concepteur (intégration de nombreux modes de génération, paramétrage facile de la génération, etc.) et nous allons montrer dans les sections suivantes plus précisément en quoi cela peut être intéressant.

8.2 Validation de la génération de trafic déterministe

Dans cette section nous montrons des résultats expérimentaux validant le fait que notre générateur de trafic est capable de rejouer, sur différentes interconnexions, le trafic d'un processeur enregistré lors d'une simulation sans interconnexion. Les conditions pour que ce résultat soit valable sont discutées dans 6.2.5.1. Ceci est une étape importante pour la validation de notre environnement de simulation, en effet il ne semble pas raisonnable de devoir, à chaque fois que l'on change l'interconnexion complète ou simplement certains paramètres de celle-ci, effectuer une nouvelle simulation avec les processeurs afin d'obtenir de nouveaux générateurs de trafic. La section 8.2.1 décrit les différentes simulations qui ont été effectuées dans ce but, et la section 8.2.3 présente et discute les résultats.

8.2.1 Simulations effectuées

Afin de montrer qu'une trace de trafic enregistrée sur une plateforme donnée peut être rejouée sans grande perte de précision dans une autre plateforme, nous avons suivi notre flot de génération de trafic (voir section 6.2 et figure 6.4), en sélectionnant une génération de trafic de type rejeu (génération de trafic déterministe). Nous avons pour cela mis en place deux plateformes de simulations :

- **DIRECT**. C'est la plateforme de collecte de la trace. Il n'y a dans ce cas pas d'interconnexion à proprement parler, le processeur et son cache sont directement reliés à une mémoire contenant toutes les informations nécessaires au déroulement du programme. Cette plateforme est illustrée sur la droite de la figure 8.1.
- **DSPIN**. Cette plateforme est beaucoup plus complète et réaliste, elle inclut différents bancs mémoires, toujours le processeur et son cache, une interface série (TTY) et un générateur de trafic (BACKTG pour *Background Traffic Generator*) ayant différentes phases afin de faire varier la latence des communications du processeur. En effet, le trafic du BACKTG va entrer en compétition avec celui du processeur, qui sera parfois mis en attente. En utilisant la possibilité de produire du trafic multiphase, on peut faire varier au cours du temps l'intensité du trafic du BACKTG, et ainsi faire varier la latence des communications du processeur. L'interconnexion que nous avons utilisée est le réseau sur puce DSPIN (voir sec-

tion 5.2.3.2). Cette plateforme est illustrée sur la gauche de la figure 8.1.

Sur chacune de ces plateformes nous avons utilisé différentes configurations :

- PROC. C'est la simulation de référence, le MIPS exécute une application.
- REJEU. Dans cette configuration on remplace le MIPS par un générateur de trafic rejouant la trace de trafic enregistrée sur la plateforme DIRECT.

Nous avons enfin utilisé les quatre applications décrites dans la section 7.3 :

- MP3. Décodage audio de 2 frames MP3.
- JPEG. Décodage d'une image JPEG de 256x256 pixels. Implémentation séquentielle de l'algorithme.
- MJPEG. Décodage d'une image JPEG de 256x256 pixels. Implémentation multithreads de l'algorithme.
- JPEG2000. Décodage d'une image J2k de 256x256 pixels.

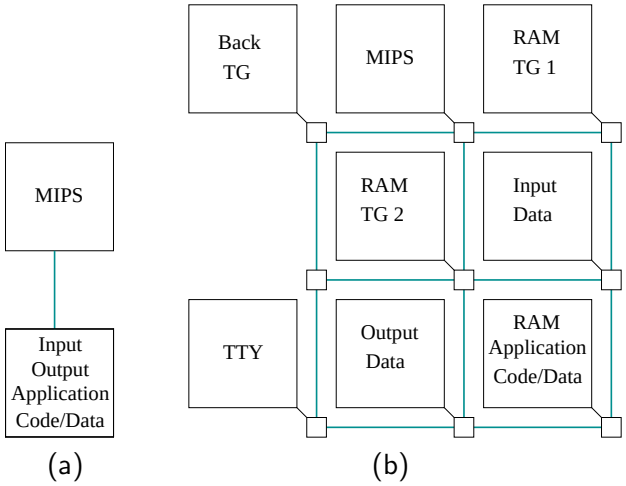


Fig. 8.1: Plateformes de simulation utilisées pour la validation de la génération de trafic déterministe : Direct (a) et Dspin (b).

Une simulation correspond donc à un triplet $(\mathscr{P},\mathscr{C},\mathscr{A})$ désignant respectivement la plateforme, la configuration et l'application. Par la suite, pour simplifier les notations, nous utiliserons $\mathscr{P}$, $\mathscr{C}$ et $\mathscr{A}$ pour dénommer respectivement l'ensemble des plateformes, des configurations et des applications. Par exemple (DIRECT,PROC,$\mathscr{A}$) correspond aux simulations avec sur la plateforme DIRECT, avec la configuration PROC pour les différentes applications. Le nombre de cycles pris par l'exécution des différentes applications sur les plateformes DIRECT et DSPIN est reporté dans le tableau 8.2.

Le flot de génération de trafic qui a été suivi implique les étapes suivantes décrites en détails dans la section 6.2.

- **Extraction de la trace de trafic de référence** à partir du fichier VCD obtenu à la sortie de la simulation (DIRECT,PROC,$\mathscr{A}$). Ceci est effectué par notre *parser*.
- **Compression et stockage** de cette trace dans des fichiers.
- **Génération du fichier de configuration** MPTG.
- **Exécution des simulations** ($\mathscr{P}$,REJEU,$\mathscr{A}$).

Plateforme	Application	Nombre de cycles
Direct	MP3	4.6 Millions
Direct	JPEG	27 Millions
Direct	MJPEG	64 Millions
Direct	JPEG2000	50 Millions
Dspin	MP3	21 Millions
Dspin	JPEG	143 Millions
Dspin	MJPEG	374 Millions
Dspin	JPEG2000	226 Millions

Tab. 8.2: Durée en nombre de cycles de l'exécution des différentes applications sur les plateformes Direct et Dspin.

- **Validation** par la comparaison de chaque simulation avec la simulation de référence (DIRECT,PROC,$\mathscr{A}$).

8.2.2 Compression

Nous discutons ici simplement les résultats de compression par l'algorithme BZ-2 des traces de trafic des différentes applications en vue d'un rejeu. Le tableau 8.3 montre la taille de ces traces ainsi que les taux de compression obtenus pour les différentes applications.

Application	Taille	Taux de compression	Temps de compression	Temps de décompression
MP3	4.3 Mo	39x	2.5 s	0.5 s
JPEG	29.7 Mo	52.2x	16.7 s	4.2 s
MJPEG	79 Mo	63.7x	46.32 s	9.7 s
JPEG2000	45.9 Mo	20x	28.23 s	5 s

Tab. 8.3: Taille et taux de compression des traces de trafic des différentes applications.

Les résultats montrent un taux de compression compris entre 20 et 60, ce qui permet d'obtenir des traces de taille raisonnable pouvant être stockées. Le temps de compression est important, mais il n'est effectué qu'une unique fois. La décompression quant à elle est assez rapide pour être négligeable devant le temps d'une simulation incluant un NOC.

8.2.3 Résultats et commentaires

Afin de pouvoir évaluer la *différence* entre le trafic produit par notre générateur de trafic et le trafic de référence (simulations $S1 = (\mathscr{P},\text{PROC},\mathscr{A})$ et $S2 = (\mathscr{P},\text{REJEU},\mathscr{A})$), nous avons défini différents indicateurs :

- C_{err}. C'est la différence relative (en pour cent) entre le nombre de cycles de $S1$ (noté $c1$) et le nombre de cycle de $S2$ (noté $c2$), soit $|c2-c1|/c1*100$. Cette métrique permet d'avoir une erreur globale, mais ne permet pas de voir si dans les détails, les deux simulations $S1$ et $S2$ sont similaires.
- D_{err}. C'est l'erreur moyenne sur le délai. Elle est calculée, si N est le nombre de transaction, $D1(k)$ le délai de la $k^{\text{ième}}$transaction dans la simulation $S1$ et $D2(k)$ le délai de la

$k^{\text{ième}}$transaction dans la simulation $S2$:

$$D_{err} = \frac{1}{N} \sum_{k=1}^{N} \frac{|D2(k) - D1(k)|}{D1(k)} * 100$$

Elle est exprimée en pour cents.

Il est inutile de regarder la différence entre les séquences d'adresse, de commande et de taille, puisqu'elles ne peuvent être différentes (voir section 6.2.4).

Plateforme	Application	C_{err}	D_{err}
Dspin	MP3	0.08 %	1.152 %
Dspin	JPEG	0.1 %	0.243 %
Dspin	MJPEG	0.02 %	1.127 %
Dspin	JPEG2000	0.01 %	0.205 %

Tab. 8.4: Résultats de précision de la génération de trafic en mode rejeu.

Les résultats sont rassemblés dans le tableau 8.4. L'erreur sur la plateforme DIRECT (non indiquée dans le tableau) est nulle pour toute les applications, ce qui est normal puisque la trace a été enregistrée sur cette plateforme. Lors du passage de ce rejeu sur la plateforme DSPIN, une légère erreur apparaît, due aux placements temporels des écritures qu'il est difficile de maintenir comme cela a été discuté dans la section 6.2.4. Néanmoins cette erreur reste très faible (inférieure à 2%), et nous la considérons comme tout à fait acceptable pour valider la capacité de notre générateur de trafic à être utilisé de manière identique sur une interconnexion quelconque, à partir d'une configuration dérivée d'une simulation sans interconnexion. D'autre part, l'erreur constatée est comparable aux résultats de [98] qui visent ce même objectif, mais pas avec les mêmes moyens (voir section 5.4.1).

Nous pouvons donc affirmer grâce à ces simulations que nous pouvons avoir confiance dans le trafic produit par notre générateur de trafic, et ce même si la configuration du réseau est modifiée. Ce point est crucial pour permettre au concepteur de tester différentes configurations de NOC, voir même différents NOC, sans avoir à retoucher aux fichiers de configuration du générateur de trafic ayant été obtenus à partir d'une simulation de référence exécutée une seule fois. Ceci permet aussi à des concepteurs d'IP de diffuser un générateur de trafic à la place du composant, évitant ainsi des problèmes de propriété intellectuelle.

Cette étape de validation est importante car dans les sections suivantes, nous allons nous intéresser à la génération de trafic stochastique sur puce, et il est déterminant, pour pouvoir analyser correctement les résultats, d'avoir confiance dans le trafic produit par notre générateur.

8.3 Segmentation automatique en phase

Cette section présente des résultats expérimentaux concernant l'algorithme de segmentation du trafic en phases introduit dans la section 6.2.3.2. La section 8.3.1 rappelle rapidement l'algorithme de segmentation et introduit quelques notations utiles. Les sections 8.3.2, 8.3.3, 8.3.4 et 8.3.5 discutent respectivement du choix des composantes, des statistiques, du nombre de phases et de la taille des blocs lors de la segmentation.

8.3.1 Rappel de l'algorithme

Nous rappelons ici rapidement les bases de l'algorithme de segmentation présenté dans la section 6.2.3.2. Étant donné une trace, c'est à dire une séquence de transactions $\{T[n]\}_{n\in\mathbb{N}}$, nous allons chercher à identifier des morceaux de cette trace ayant un comportement similaire et ceci implique les étapes suivantes :

1. **Division en blocs**. La séquence est tout d'abord divisée en blocs de taille fixe.
2. **Vecteurs de bloc** (BV). Pour chacun de ces blocs, nous allons définir un vecteur le caractérisant. Comme nous souhaitons identifier les phases ayant un comportement statistique similaire (stationnaire sur ces blocs), ce vecteur sera composé de différentes informations de ce type, comme la moyenne, la variance, l'histogramme empirique (voir section 1.4.2) ou la covariance.
3. **Segmentation**. Nous utilisons ensuite l'algorithme de classification *k-means*, qui va associer à chaque bloc (à chaque vecteur en fait), un identifiant. Cette association est basée sur une distance (voir section 6.2.3.1).
4. **Résultats**. L'identifiant correspond au numéro de la phase, c'est à dire que les blocs ayant le même identifiant appartiennent à la même phase. Nous pouvons donc obtenir une liste des phases, chaque phase étant caractérisée par une liste de blocs.

Il reste maintenant à définir les vecteurs de blocs. Cette étape laisse un large choix, puisque l'on peut décider d'utiliser différentes statistiques (moyenne, variance, histogramme empirique et covariance) calculées sur une ou plusieurs composantes de la séquence de transaction $T(k)$. On peut par exemple décider de segmenter en utilisant le délai moyen, et dans ce cas le vecteur caractéristique d'un bloc sera la moyenne du délai sur le bloc. Ceci revient à, si $\mathbf{B_i}$ est le BV du $i^{\text{ième}}$bloc, et si M est la taille des blocs :

$$\mathbf{B_i} = \Big(\frac{1}{M}\sum_{k=Mi}^{k=M(i+1)} D(k)\Big)$$

Si on décide d'utiliser la moyenne sur le délai et la taille, alors le BV sera un vecteur à 2 composantes :

$$\mathbf{B_i} = \Big(\frac{1}{M}\sum_{k=Mi}^{k=M(i+1)} D(k), \quad \frac{1}{M}\sum_{k=Mi}^{k=M(i+1)} S(k)\Big)$$

La segmentation implique donc de nombreux paramètres, et on peut identifier une segmentation particulière $\mathscr{S}$ par le vecteur de ces paramètres :

$$\mathscr{S} = (\mathscr{C}, \mathscr{T}, M, N)$$

avec :

- $\mathscr{C}$ représente l'ensemble des composantes utilisées (voir section 8.3.2).
- $\mathscr{T}$ représente l'ensemble des statistiques sélectionnées (voir section 8.3.3).
- M est la taille des blocs (voir section 8.3.5).
- N est le nombre de phase (voir section 8.3.4).

Nous ne discuterons pas ici les différentes distances pouvant être utilisées (nous avons utilisé la distance euclidienne, voir section 6.2.3.2), cela fait partie des perspectives de recherche.

Chacun de ces paramètres est discuté dans les sections suivantes. Nous présenterons ici des résultats sur la segmentation de la trace correspondant à la simulation (DIRECT,PROC,MP3) (voir section précédente et figure 8.1) uniquement. En effet, la segmentation doit être effectuée sur une trace de référence, ce qui justifie l'utilisation de la configuration MIPS et de la plateforme DIRECT. Les traces de trafic pour l'exécution des applications JPEG et MJPEG ne présentent pas plusieurs phases, elle peuvent être modélisées entièrement car elles sont raisonnablement stationnaires. En revanche, dans le cas de l'application MP3, comme cela est illustré sur la figure 6.2, la trace n'est pas stationnaire, et cette trace constitue un bon candidat pour tester et valider notre méthode de segmentation. La trace de l'application JPEG2000 présente aussi différentes phases, nous ne présentons néanmoins pas ces résultats ici car ils sont similaires à ceux obtenus avec l'application MP3.

8.3.2 Sélection des composantes

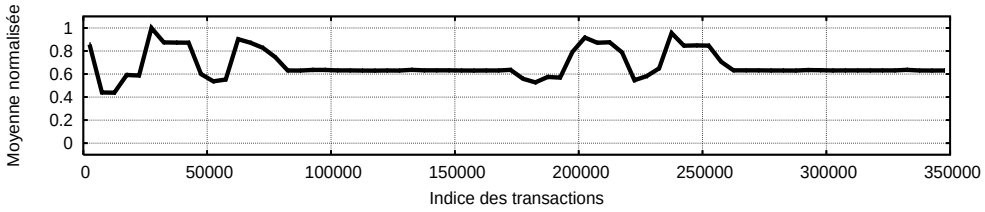
Dans cette section, nous cherchons à évaluer quelles composantes doivent être utilisées pour effectuer la segmentation. Pour cela, nous avons fait varier $\mathscr{C}$ et fixé $\mathscr{T}$ = {Moyenne}, $N = 4$ phases et $M = 5000$ transactions.

La Figure 8.2 montre l'évolution de la moyenne normalisée (rapportée entre 0 et 1) des différentes composantes de la séquence $\{T[n]\}_{n\in\mathbb{N}}$ (délai, taille et commande et adresse) calculée sur des blocs consécutifs disjoints de taille $M = 5000$ transactions. Cette figure permet de voir que ces trois évolutions sont corrélées, c'est à dire qu'un changement de comportement se répercute dans toutes les composantes. Ceci est *a priori* une remarque en faveur du fait que les segmentations effectuées avec une de ces trois composantes devraient donner des résultats similaires.

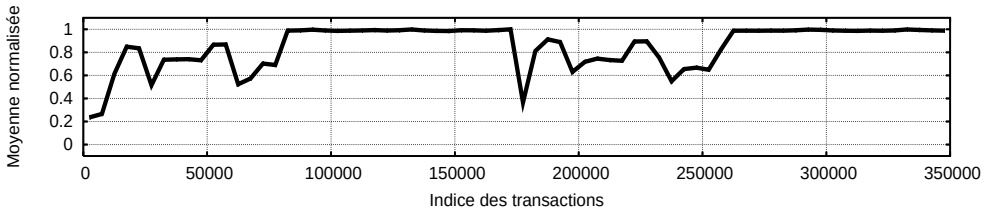
Nous avons effectué quatre segmentations différentes en utilisant quatre BV différents de dimension 1 (moyenne des délais, des tailles, des commandes et des adresses), et le résultat de ces segmentations est représenté sur la figure 8.3 (la première ligne montre l'évolution du délai). On peut tout d'abord voir dans la figure 8.3 que l'algorithme de segmentation arrive à identifier les différentes phases dans le trafic. En particulier, l'application MP3 décode dans ces simulations deux trames, le traitement de chaque trame étant similaire. L'algorithme arrive à identifier ces deux traitements, et à identifier différentes phases à l'intérieur de chacun. On remarque aussi que les premiers blocs, correspondant à la période de démarrage (*boot* du système d'exploitation), sont dans des phases particulières. Cela est dû au fait que pendant cette période, les communications sont particulières puisque différentes sections mémoires sont initialisées (longues séquences d'écritures). Le période de démarrage pénalise donc un peu l'algorithme car ces phases n'apparaissent plus ensuite. On pourrait ne pas tenir compte des premiers blocs, mais notre objectif n'est pas ici de selectionner les parties intéressantes de la trace de trafic, mais simplement de toutes les identifier.

Couple	MP3	JPEG2000	JPEG	MJPEG
Delai - Commande	0.29	-0.12	0.17	-0.15
Delai - Taille	-0.29	0.12	-0.17	0.15
Delai - Adresse	0.11	0.034	0.058	-0.044
Commande - Taille	-1	-1	-1	-0.99
Commande - Adresse	0.37	0.28	0.56	0.26
Taille - Adresse	-0.37	-0.28	-0.56	-0.23

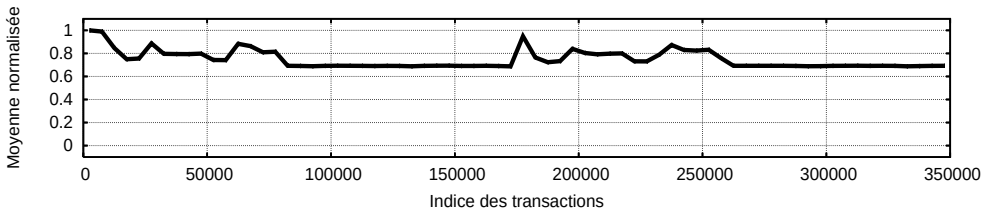
Tab. 8.5: Coefficient de corrélation entre les différentes composantes de la séquence de transactions.



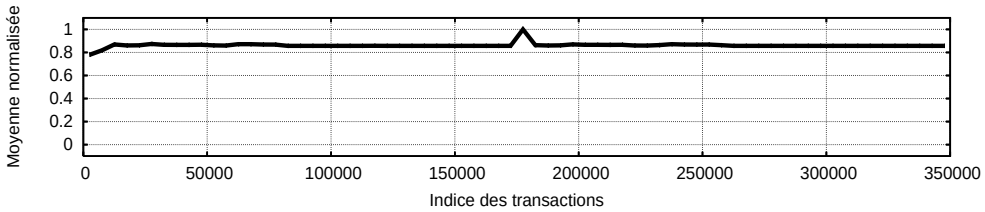
(a) Délai moyen



(b) Taille moyenne



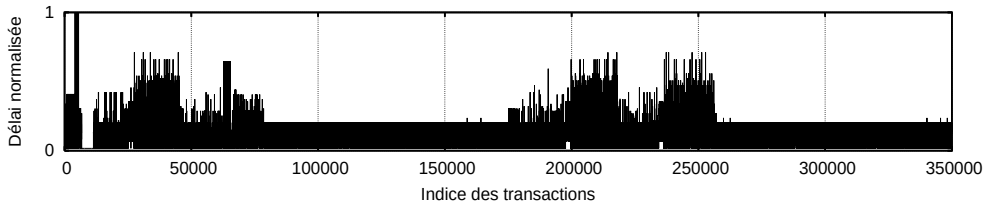
(c) Commande moyenne



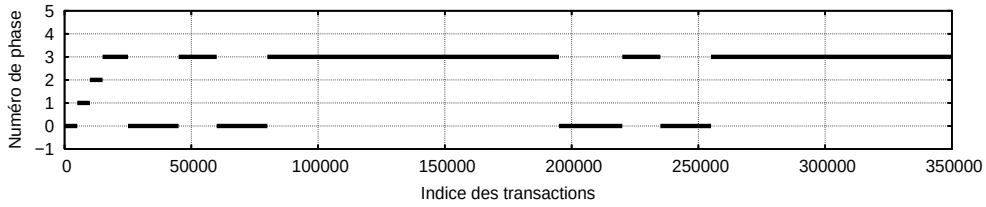
(d) Adresse moyenne

Fig. 8.2: Évolution de la moyenne normalisée de différentes composantes de la séquence de transactions émise par le processeur MIPS exécutant l'application décodage audio MP3.

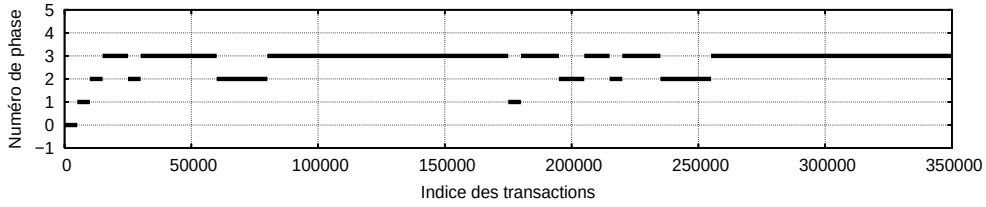
On peut observer, en comparant les figures 8.3(b), (c), (d) et (e), que les segmentations obtenues sur la taille et la commande sont similaires, et que celles effectuées sur le délai et l'adresse sont légèrement différentes. Ceci peut s'expliquer par la présence de corrélations entre les différentes composantes de la séquence de transactions. Le *coefficient de corrélation* introduit dans la section 1.1.4 permet de quantifier ces corrélations. Il varie entre -1 (corrélation négative) et 1 (corrélation positive) en passant par 0 (aucune corrélation). Le tableau 8.5 montre ce coefficient pour chaque couple de composantes et différentes applications. On note que la commande et la taille sont directement corrélées. Ceci s'explique par le fait que la taille des requêtes est directement liée à la commande (une lecture fait la taille d'une ligne de cache, et une écriture fait un mot puisqu'il n'y a pas de tampon d'écritures postées dans le cache). La commande et la taille sont un peu corrélées avec l'adresse puisque certaines parties de la mémoire ne sont accédées qu'en lecture (le flot d'instructions par exemple), et d'autres uniquement en écriture (une interface série par exemple). Le délai est quand lui relativement peu corrélé avec les autres composantes. Ceci justifie d'ailleurs la séparation entre le placement temporel des transactions et le contenu de ces dernières. Il est important de noter que ces remarques sont spécifiques au trafic des processeurs, une nouvelle étude serait à faire avec un trafic émis par un accélérateur matériel car la corrélation entre commande et taille des transactions par exemple n'aurait pas de raison d'être



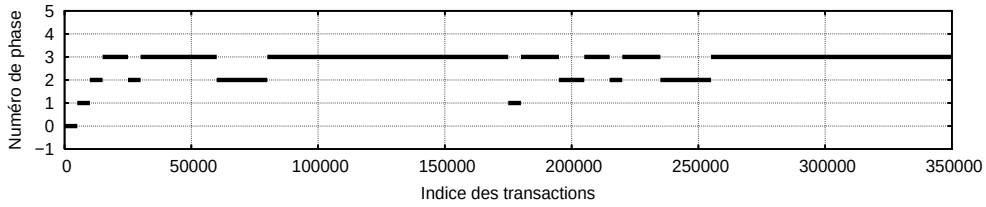
(a) Trace de délai moyen



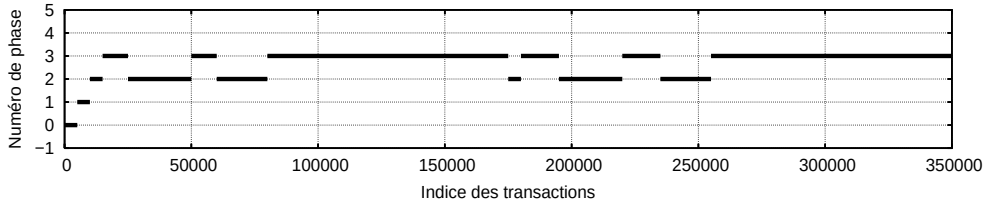
(a) Délai moyen



(b) Taille moyenne



(c) Commande moyenne

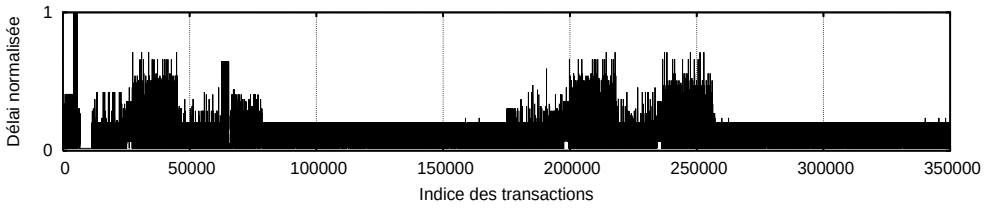


(d) Adresse moyenne

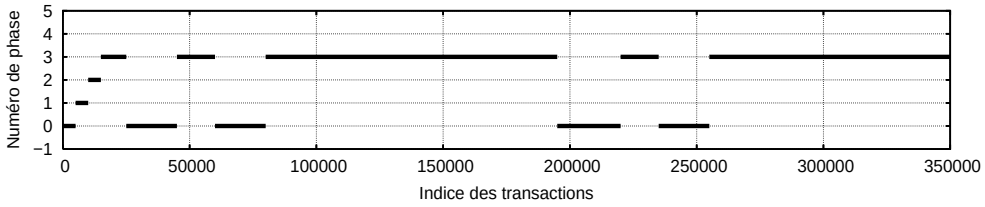
Fig. 8.3: Différentes segmentations obtenues pour différents vecteurs de blocs.

si forte.

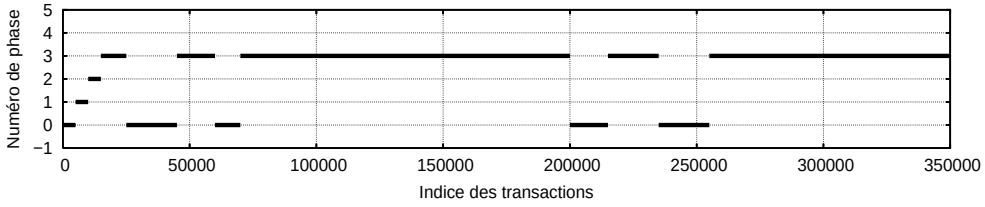
En conclusion, on peut dire que la matrice des coefficients de corrélations entre les différentes composantes permet d'avoir une information quantitative pouvant aider à la selection des composantes. En effet, si deux composantes sont très corrélées (comme la taille et la commande dans les résultats présentés ici), alors il n'est pas nécessaire d'intégrer les deux au vecteur de bloc. D'une manière générale, la section des composantes dépend de l'utilisation qui sera faite de la segmentation, l'utilisation d'une seule composante rendra uniquement compte des phases dans l'évolution de cette composante particulière. L'utilisation de plusieurs composantes offre un bon compromis comme cela sera montré dans la section 8.4.



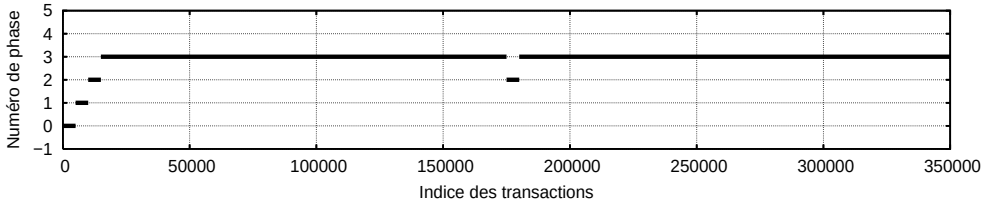
(a) Moyenne du délai



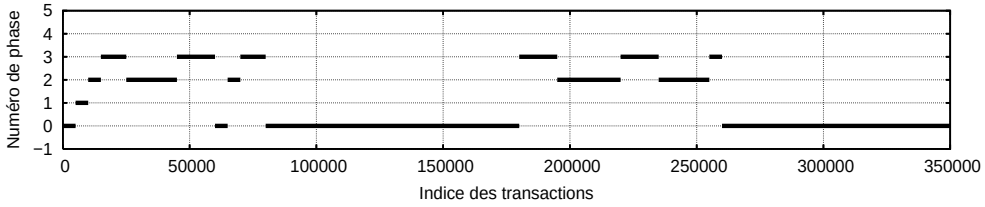
(b) Moyenne



(c) Variance



(d) Histogramme empirique



(e) Covariance

Fig. 8.4: Différentes segmentations utilisant différentes statistiques calculées sur les délais (moyenne, variance, histogramme empirique et covariance).

8.3.3 Sélection des statistiques

Dans cette section, nous explorons la segmentation faite en utilisant différentes statistiques. Nous avons donc fait varier $\mathscr{T}$, et fixé $\mathscr{C} = \{Delai\}$, $N = 4$ et $M = 5000$ transactions.

La figure 8.4 montre que la segmentation en utilisation la moyenne ou la variance donne des résultats similaires (figure 8.4(b) et (c)). L'utilisation de l'histogramme empirique (voir section 1.2.2) et de la covariance (figure 8.4(d) et (e)) permet d'identifier les deux grandes phases correspondant aux deux frames décodées successivement, mais le découpage obtenu est différent. Au vue de l'évolution des différentes composantes (figure 8.4(a) et 8.3), on peut néanmoins

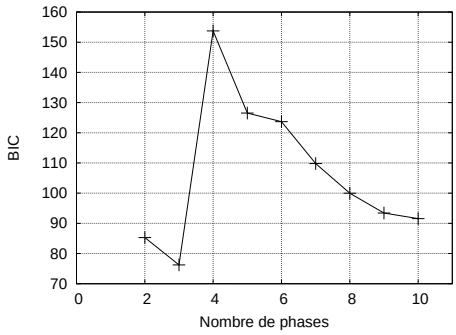


Fig. 8.5: Évolution du BIC en fonction du nombre de phases (segmentation du la moyenne des délais).

remarquer que les phases identifiées avec la moyenne et la variance sont plus proches des phases de trafic manuellement observées dans le trafic que celle identifiées avec l'histogramme empirique ou la covariance. Nous avons donc décidé d'utiliser une combinaison de la moyenne et de la variance (vecteur de bloc à deux composantes, la moyenne et la variance de chaque bloc).

8.3.4 Nombre de phases

Nous explorons ici le nombre de phases N, avec $\mathscr{T}$ = {Moyenne}, et fixer $\mathscr{C} = \{Delai\}$ et $M = 5000$ transactions.

L'algorithme de classification *k-means* fonctionne pour un nombre de classes (de phases dans notre cas) N donné. Afin de pouvoir décider quel nombre de phases utiliser pour obtenir une bonne segmentation, nous avons mis en place la même méthode que celle de Calder *et al.* [138], c'est-à-dire que nous allons exécuter plusieurs fois l'algorithme *k-means* pour différentes valeurs de N (entre 3 et 10 en pratique) et regarder l'évolution du BIC en fonction de N. On sélectionnera alors le plus petit nombre de phases telle que le BIC soit au moins égal à 80 % du BIC maximum, ceci pour essayer de ne pas garder un nombre de phases trop élevé si ce n'est pas absolument nécessaire.

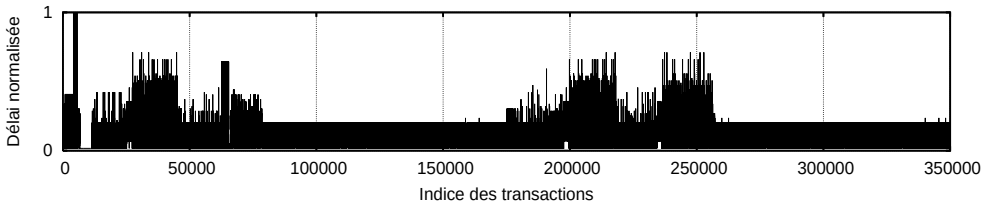
La figure 8.6 montre une segmentation effectuée sur la moyenne des délais pour différents nombres de phases (en haut l'évolution du délai, puis les segmentations obtenus pour différents nombres de phases). La figure 8.5 montre quant à elle le BIC en fonction du nombre de phases.

Nous disposons ainsi, par l'intermédiaire du BIC, d'un indicateur permettant de choisir le nombre de phases, qui est ici quatre puisque c'est la segmentation qui possède le BIC le plus haut, et que le BIC pour des nombres de phases inférieurs est trop petit.

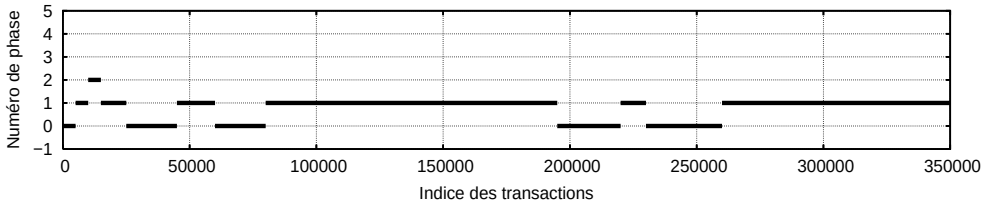
8.3.5 Taille des blocs

Pour finir l'exploration des différents paramètres de la segmentation, nous avons fait varier la taille M des blocs, en gardant les autres paramètres constants : $\mathscr{T} = \{Moyenne\}, \mathscr{C} = \{Delai\}$ et $N = 4$ phases.

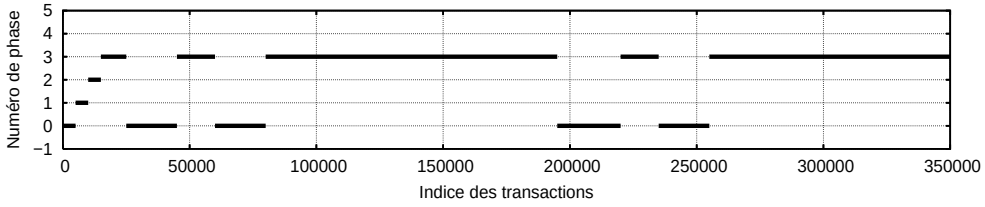
Le choix de cette taille est un compromis entre une meilleure estimation des statistiques (il faut pour cela beaucoup d'échantillons, donc une taille importante) et une meilleure résolution (il faut pour cela des blocs de petites tailles, afin d'en avoir beaucoup et de pouvoir détecter les changements de phases plus précisément).



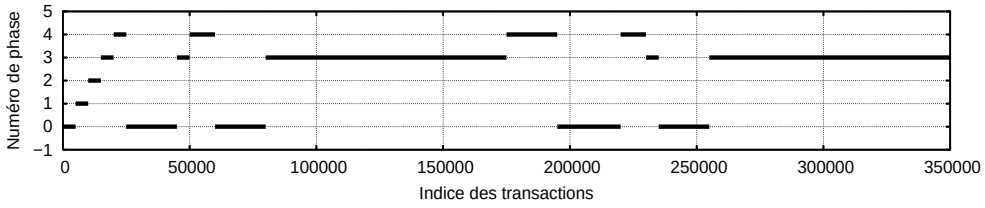
(a) Trace originale



(b) 3-means



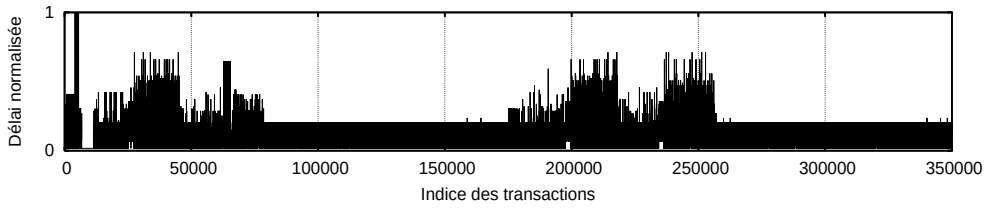
(c) 4-means



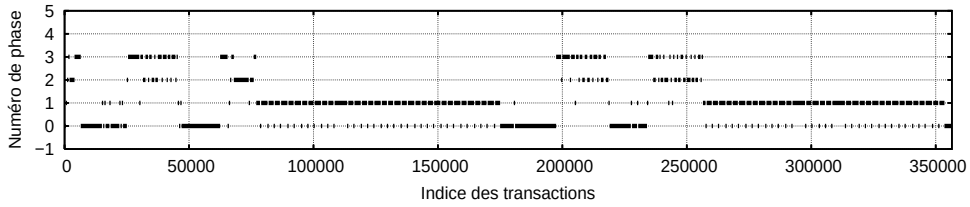
(d) 5-means

Fig. 8.6: Segmentation pour différents nombres de phases, obtenue avec une segmentation basée sur la moyenne des délais.

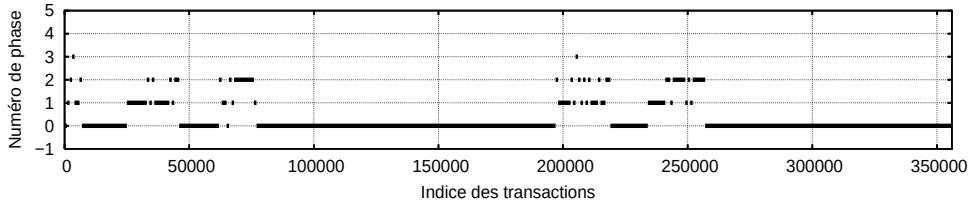
Nous avons donc effectué différentes segmentations (en utilisant la moyenne des délais sur l'application MP3) et les résultats sont présentés sur la figure 8.7. Comme sur les précédentes figures, la figure 8.7-(a) montre la trace de délai moyen, et les courbes suivantes montrent la segmentation pour différentes tailles de blocs. On voit bien que si la taille est trop petite (figure 8.7-(a)), alors la segmentation obtenue est plus hachée, et ceci est dû au fait que l'évaluation des statistiques sur des blocs de petite taille n'est pas très performante, à cause de la convergence des estimateurs. En effet, la loi des grands nombres (voir section 1.1.2), sur laquelle tous ces estimateurs sont basés, est valable quand le nombre d'observations est grand. Lorsque la taille des bloc est trop grande, en revanche, on a de grandes chances pour que des changements de phases interviennent au milieu d'un bloc, ce qui rendent les résultats moins exploitables, car moins représentatif des les phases réellement présentes dans le trafic. On observe donc que les résultats offrant un meilleur compromis sont pour $M = 1000$ et $M = 5000$. Ces valeurs sont néanmoins reliées à la nature de l'application ainsi qu'à l'utilisation faite de ce découpage en phases. Nous n'avons pas étudié encore de moyen pour automatiser ou pour fournir des indicateurs pour aider au choix de ce paramètre. En effet, le BIC, utilisé pour déterminer le nombre de phases ne peut pas être utilisé ici car on ne peut pas directement comparer sa valeur entre deux segmentations ayant un nombre de blocs différent. Ceci fait partie des perspectives de recherche.



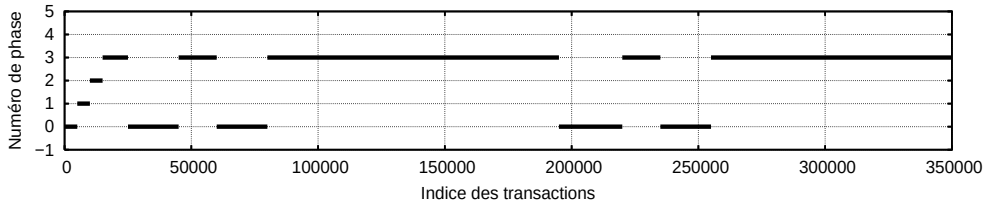
(a) Trace originale



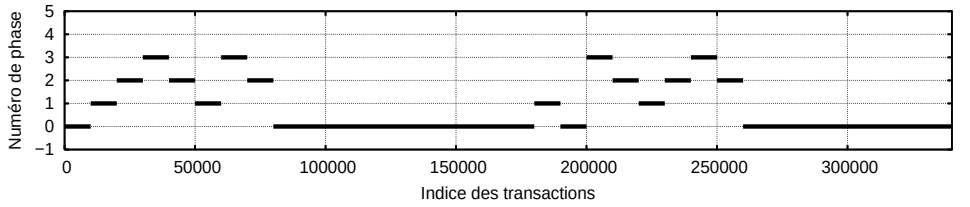
(b) $M = 500$



(c) $M = 1000$



(d) $M = 5000$



(e) $M = 10000$

Fig. 8.7: Segmentation pour différentes taille de bloc, obtenue avec une segmentation basée sur la moyenne des délais.

8.3.6 Conclusion

Nous avons dans cette section présenté une exploration de chaque paramètre de la segmentation en phases d'une trace de référence. Nous avons pu ainsi déterminer les différents paramètres (composantes, statistiques, tailles des blocs et nombre de phases) dans le cas particulier de l'application de décodage audio MP3. Cette étape nous a permis d'identifier les décisions à prendre et de fournir dans certains cas des moyens pour aider à prendre ces décisions. Cela nous permet d'être confiant pour une future automatisation du choix de ces paramètres, qui est une perspective de recherche.

Une fois ces choix effectués, l'algorithme de segmentation automatique que nous avons uti-

lisé fournit un découpage en phases de trafic qui peut être utile au concepteur de réseau. Ce découpage va aussi nous servir de base à la génération de trafic stochastique multiphase qui fait l'objet de la section suivante.

8.4 Génération de trafic stochastique multiphase

Nous montrons dans cette section des résultats concernant l'utilisation de l'aspect multiphase exploré dans la section précédente. En effet, à partir de la segmentation, nous pouvons introduire le concept de *génération de trafic stochastique multiphase*. Cela signifie que la procédure d'estimation des paramètres des modèles stochastiques sera effectuée indépendamment sur chaque phase. La sélection des modèles reste manuelle. Dans cette section, nous montrons des résultats en terme de précision de cette technique de génération.

L'objectif de cette génération de trafic est de combiner les avantages d'une génération de trafic tenant compte du comportement du composant que le TG (comme la génération de trafic déterministe, voir section 5.4.1) et d'une génération de trafic à base de réalisations de processus stochastiques (comme la génération d'une charge aléatoire sur un réseau). Chacun de ces deux modes de générations a ces avantages propres (voir section 5.4), les combiner repose sur deux étapes importantes :

- **Segmentation en phases** de la trace de référence.
- **Estimation des statistiques** d'ordre un et deux sur chacune des phases identifiées.

Afin de montrer l'intérêt de ce mode de génération, nous avons utilisé les deux plateformes DIRECT et DSPIN comme pour l'évaluation de la génération de trafic déterministe (voir section 8.2). Sur ces plateformes, nous avons utilisé les configurations suivantes :

- **PROC**. C'est la configuration de référence avec les MIPS exécutant une application (voir section 8.2). Les autres configurations seront comparées à cette configuration de référence.
- **MPTGN**. Cette configuration correspond à une génération de trafic stochastique multiphase avec N phases. Nous avons, pour chaque phase, enregistré l'histogramme empirique (celui-ci ne correspondant à aucune loi standard de manière satisfaisante), et n'avons pas considéré la covariance (processus IID, voir section 1.3.1).
- **REJEU**. C'est la configuration de rejeu (voir section 8.2).
- **STG** (pour *Simple TG*). Cette configuration correspond à une génération à débit constant, avec des destinations uniformément réparties. Cela va nous permettre de comparer nos résultats avec une génération stochastique sans procédure d'estimation. Pour que cette comparaison soit juste, nous avons fixé le débit égal au débit moyen de la trace de référence.

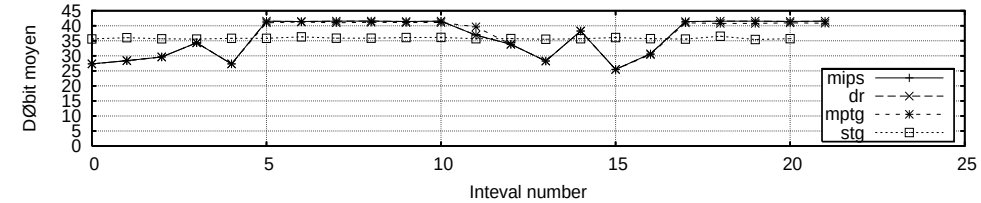
Les résultats ne sont présentés ici que sur l'application MP3 qui présente un comportement où des phases ont pu être identifiées. A la suite des discussions poursuivies dans la section précédente, nous avons utilisé une segmentation basée sur la moyenne et la variance des délais et des tailles, et la taille M des blocs a été fixée à 5000 transactions (voir section 8.3).

Étant donné que la génération de trafic stochastique est aléatoire par nature, il n'est pas possible d'utiliser les métriques définies dans la section 8.2 pour comparer le trafic de référence avec le trafic stochastique car on ne peut plus comparer les échantillons *deux-à-deux*. Nous avons donc défini de nouvelles métriques permettant de rendre compte de la *ressemblance* entre trafic produit par la simulation de référence $S1$ et le trafic produit par une simulation avec des gé-

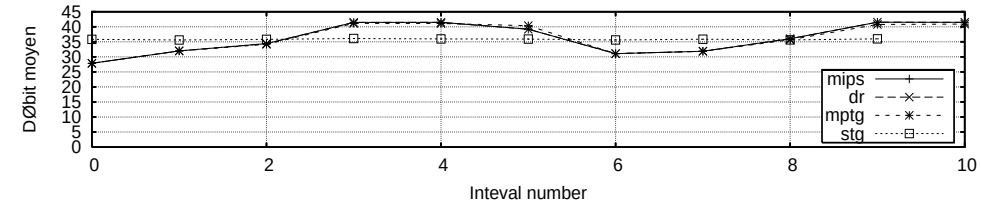
nérateurs de trafic $S2$. Ces métriques visent à comparer l'évolution de la moyenne de chaque composante (délai, taille, adresse, commande et débit), à une *échelle* notée L donnée. L'échelle L est la granularité de l'analyse, la moyenne de chaque composante sera calculé par blocs consécutifs disjoints de taille L transactions. Des exemples de telles évolutions sont représentées sur la figure 8.8 pour différentes valeurs de L.

Formellement, si $M_{S1}(i)$ dénote la moyenne d'une composante $\mathscr{C}$ calculée dans le bloc i (transactions dont l'index est compris entre iL et $(i+1)L$) pour la simulation $S1$, si $M_{S2}(i)$ représente la même chose pour la simulation $S2$ et si enfin n dénote le nombre de blocs, alors on définit l'erreur (en pour cent) pour la composante $\mathscr{C}$, notée $E_{\mathscr{C}}$:

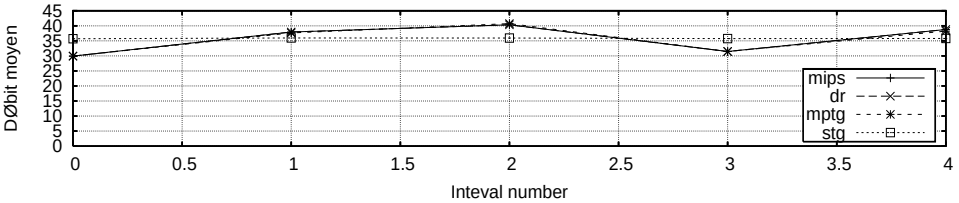
$$E_{\mathscr{C}}(L) = \frac{1}{n}\sum_{i=1}^{n} |M_{S2}(i) - M_{S1}(i)|/M_{S1}(i) * 100.$$



(a) $L = 100000$ cycles)



(b) $L = 200000$ cycles)

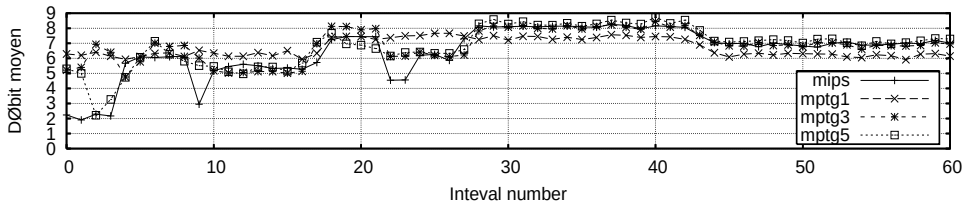


(c) $L = 400000$ cycles)

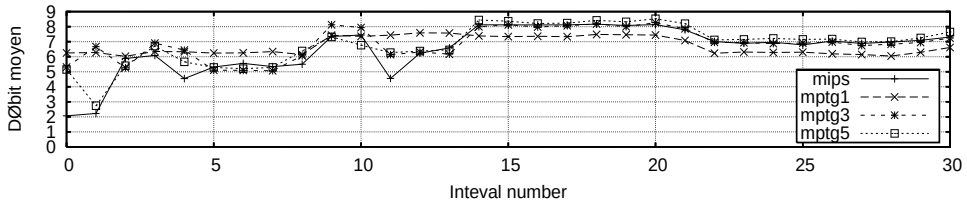
Fig. 8.8: Évolution du débit moyen des différentes configurations sur la plateforme Direct pour trois échelles L différentes : 1000000 cycles (a), 200000 cycles (b) et 400000 cycles (c).

Les figures 8.8 et 8.9 montrent l'évolution du débit moyen des différentes composantes en fonction de l'échelle. Cette évolution est une ligne droite pour La configuration MPTG1 pour la plateforme DIRECT car il n'y a qu'une seule phase. La sorte d'oscillation apparaissant sur la simulation sur la plateforme DSPIN est induite par le composant BACKTG (voir section 8.2.1), qui introduit une latence variable et donc un débit variable dans la simulation. Cette oscillation est d'ailleurs présente pour toutes les configurations.

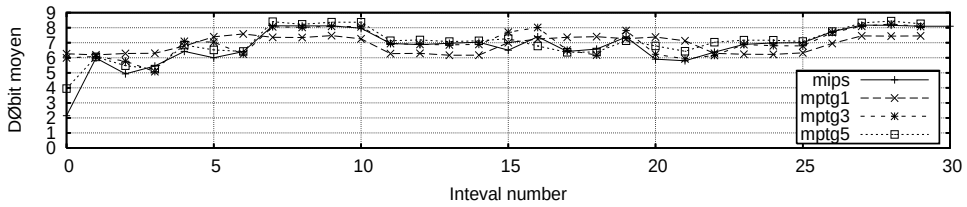
On peut voir sur ces graphiques que les configurations MPTG3 et MPTG5 *suivent* bien le comportement du trafic de référence, c'est à dire que les courbes d'évolution du débit sont très proches de la courbe de référence (PROC). Ce n'est pas le cas des configurations MPTG1 et STG, qui ne reproduisent pas les phases de la trace de référence.



(a) $L = 100000$ cycles



(b) $L = 200000$ cycles



(c) $L = 400000$ cycles

Fig. 8.9: Évolution du débit moyen des différentes configurations sur la plateforme Dspin pour trois échelles différentes : 1000000 cycles (a), 200000 cycles (b) et 400000 cycles (c).

Config.	Delay	Size	Cmd	Thput	Latency	Cycle
Rejeu	0	0	0	0	0	0
STG	31.78	34.31	6.91	13.64	0	32.1
MPTG1	7.83	15.18	6.43	11.45	0	1.83
MPTG3	1.97	8.34	3.28	5.58	0	1.55
MPTG5	1.33	3.27	1.15	2.60	0	1.21

Tab. 8.6: Erreur relative sur différentes composantes par rapport à la simulation de référence sur la plateforme Direct.

Les métriques introduites plus haut ont été calculées, avec une échelle de $L = 100000$ cycles, pour les différentes composantes, sur les différentes plateformes et pour les différentes configuration. Elles sont toutes reportées dans les tableaux 8.6 (plateforme DIRECT) et 8.7 (plateforme DSPIN). On voit, comme cela pouvait être supposé, que l'erreur diminue avec le nombre de phases, indiquant une génération plus fidèle au trafic de référence. L'erreur de la génération stochastique multiphase est aussi toujours comprise entre celle de la configuration STG (pas de prise en compte de la trace de référence) et celle de la configuration REJEU (rejeu de la trace de référence). Il est important de noter que l'erreur relativement importante sur le délai induite par l'utilisation d'une génération stochastique se répartit au cours de la simulation. En effet, l'erreur sur le nombre de cycle totale reste toujours faible, ce qui indique que la charge de trafic moyenne sur l'ensemble de la simultion est bien reproduite par le générateur de trafic.

Ces résultats montrent que la génération de trafic stochastique multiphase est intéressante pour le dimensionnement des réseaux sur puce. Si la précision n'est pas aussi bonne qu'un re-

Config.	Delay	Size	Cmd	Thput	Latency	Cycle
Rejeu	1.15	0	0	0.20	0.12	1.7
STG	41.28	75.24	7.71	102.32	27.83	11.3
MPTG1	18.60	14.76	6.26	12.7	10	2.83
MPTG3	17.19	8.17	3.26	6.21	0.78	2.81
MPTG5	14.77	3.24	1.21	5.65	0.63	2.75

Tab. 8.7: Erreur relative sur différentes composantes par rapport à la simulation de référence sur la plateforme Dspin.

jeu déterministe, les phases du trafic de référence sont précisément simulées, ce qui représente un point important à prendre en compte lors du dimensionnement d'un NoC. Ce type de génération de trafic offre les avantages d'une génération stochastique (pas de limite de temps de génération, introduction d'aléatoire, possibilité de pousser le système dans des zones à risques, modélisation du trafic) et les avantages d'une prise en compte précise du comportement du composant que l'on souhaite émuler. Il trouve donc logiquement sa place entre une génération stochastique simple et le rejeu d'une trace de référence. Le choix entre ces différents niveaux de précision dépend de l'utilisation faite de la génération de trafic. Pour l'évaluation des NoC dans les premières étapes de la conception (choix de topologies, stratégies de routage, etc.), une génération de charge aléatoire simple suffit. En revanche, lors du dimensionnement des commutateurs, un rejeu voir une simulation des composants eux-même est nécessaire. La génération de trafic stochastique multiphase s'insère entre les deux et notre environnement permet d'établir un lien entre ces différentes types de génération de trafic. Le point clef est, selon nous, de fournir au concepteur une description du trafic en terme de phases. Il pourra en effet se servir de cette information en soi pour entrelacer les communications des différents composants intégrés dans le même SoC.

8.5 Résultats sur la longue mémoire

Dans cette section, nous présentons dans un premier temps des résultats concernant la présence de longue mémoire dans le trafic des processeurs sur puce (section 8.5.1), et dans un second temps des résultats concernant l'impact de la longue mémoire sur les performances d'un NoC (section 8.5.2).

8.5.1 Longue mémoire et trafic des processeurs sur puce

Cette section explore la présence de la caractéristique de longue mémoire dans le trafic des processeurs sur puce.

8.5.1.1 Simulations effectuées

Nous avons utilisé les traces de référence collectées sur la plateforme DIRECT (sans interconnexion, comme celle à droite de la figure 8.1), incluant un MIPS associé à son cache et une mémoire. Le processeur exécute les différentes applications présentées dans la section 7.3. Nous avons analysé les traces de débit agrégé (dans des fenêtres de taille $\Delta = 100$ cycles), et utilisé la représentation de la covariance en échelles (diagramme LD, voir section 2) pour évaluer la présence de longue mémoire. Celle-ci se caractérise dans ces diagrammes par un comportement

linéaire dans la limite des grandes échelles, la pente de la droite étant directement reliée au paramètre de longue mémoire H. Une comportement IID (aucune mémoire), se matérialise par une droite horizontale.

De par la présence de phases dans le trafic mis en évidence dans la section précédente, nous n'avons pas analysé les traces complètes, nous avons au contraire recherché dans ces traces des morceaux raisonnablement stationnaires. Cette étape a été effectuée manuellement au début, puis à l'aide de l'algorithme de segmentation présenté dans la section précédente.

8.5.1.2 Résultats

Pour certaines applications et certains morceaux de trace, nous avons pu identifier de la longue mémoire, néanmoins le résultat principal est que pour la majorité du trafic que nous avons analysé, la longue mémoire n'a pas pu être identifiée de manière claire. On peut donc affirmer que la longue mémoire n'est pas une propriété présente de manière évidente dans le trafic que nous avons analysé, ce qui n'est pas le cas du trafic Internet par exemple, ou cette caractéristique est systématiquement retrouvée. Ce processeur étant assez représentatif de ceux couramment intégrés dans les puces, il est probable que ce résultat s'étende aux autres processeurs embarqués. Pour être plus précis, des observations particulières par application sont données ci-après :

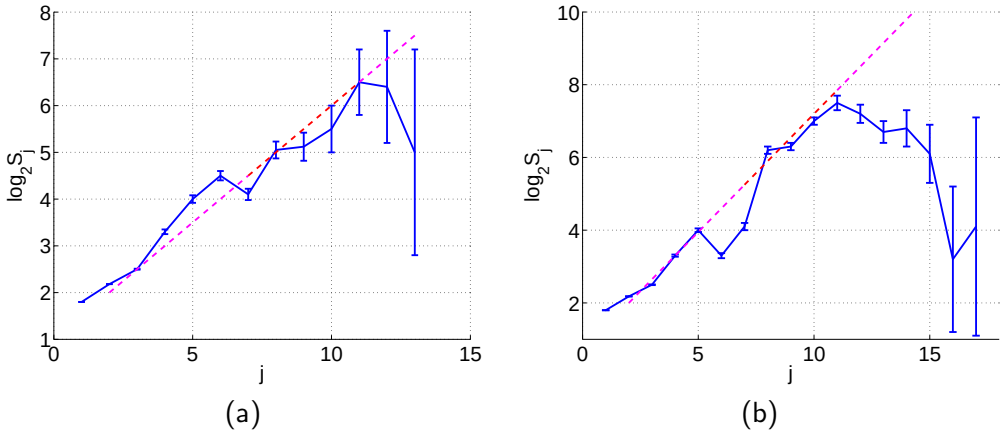


Fig. 8.10: Diagramme LD du trafic issu d'un processeur exécutant l'application MJPEG, pour une image de 64x64 (a) et une image de 256x256 (b).

- **MP3**. Décodage audio de 2 frames MP3. Dans ce trafic, on peut affirmer que la longue mémoire est absente.
- **JPEG**. Décodage d'une image JPEG de 256x256 pixels. Pour cette implémentation, aucune trace de longue mémoire n'a pu être identifiée, nous avons pourtant pu identifier de grande parties stationnaires.
- **MJPEG**. Décompression d'une image d'un flux MJPEG. Il s'agit d'une implémentation multithread de l'algorithme. Le trafic de cette application est remarquablement stationnaire, et nous avons pu identifier de la longue mémoire comme cela est illustré sur la figure 8.10(a). On voit sur cette figure se dégager un comportement linéaire, que l'on peut assimiler à de la longue mémoire avec un paramètre de Hurst estimé à $H = .85$. Néanmoins, en effectuant des simulations plus longues (décodage d'une image 256x256), la longue mé-

moire est beaucoup moins évidente comme le montre la figure 8.10(b). En effet, on assiste bien à une croissance linéaire sur la gamme d'échelles $2^1 - 2^1 1$, mais dans les grandes échelles ce comportement n'est pas maintenu. On peut donc en conclure que la longue mémoire est présente sur cette gamme d'échelle, ce qui signifie tout de même que le trafic est fortement auto-corrélé sur des intervalles de $2^1 1 \sim 200000$ cycles. Il est important aussi de noter que les caractéristiques statistiques d'ordre 2 de ce trafic sont très différentes de l'implémentation séquentielle de JPEG. Cela peut être imputé au système d'exploitation qui a, dans le cas d'une application multithread, un rôle plus bien important que pour une application séquentielle. Ceci introduit des communications, et se traduit par des caractéristique statistiques différentes.

- **JPEG2000**. Décodage d'une image j2k de 256x256 pixels. Pour certaines phases de cette application, on a pu identifier de la longue mémoire. La figure 8.11 (a) montre le diagramme LD de l'un de ces morceaux, la valeur du paramètre H estimée est de 0.77. Ces morceaux ont été identifiés dans une partie spécifique de l'algorithme (décodeur entropique), les autres parties ne présente pas cette caractéristique.

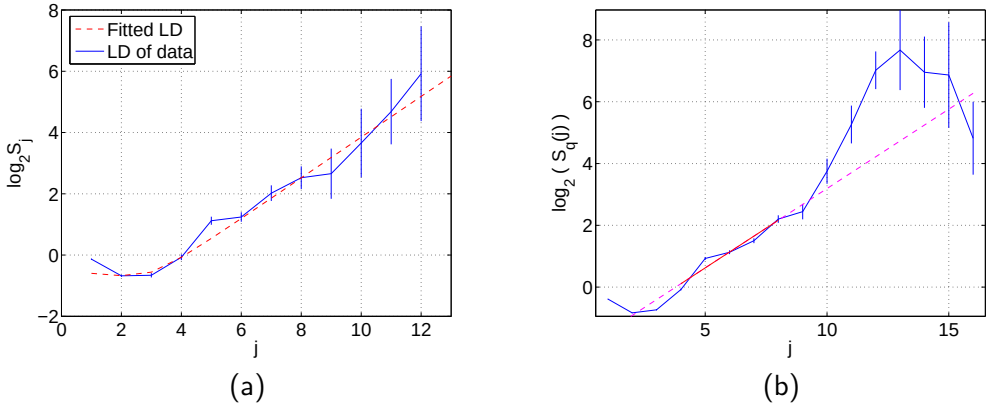


Fig. 8.11: Diagramme LD du trafic issue d'un processeur exécutant l'application j2k, pour un morceau de la trace (a) et l'ensemble de celle-ci (b).

En conclusion, on peut affirmer que, à quelques exceptions près, la longue mémoire n'est pas présente de manière significative dans le trafic des processeurs sur puce. Ceci tend à montrer que la répartition des défauts de cache dans le temps ne présente pas de longue mémoire, puisque le trafic que nous analysons est, en grande partie, composé de ces défauts (le reste étant les accès mémoires dans les espaces non-cachés). Ce résultat était en quelque sorte prévisible, puisqu'il n'existe pas, dans la littérature, d'étude tendant à montrer que la répartition des défauts de cache dans le temps présente de la longue mémoire (il n'existe pas à notre connaissance d'étude montrant le contraire ceci dit). Nous avions, au départ, intégré la prise en compte de cette caractéristique dans le but d'analyser du trafic issu d'accélérateurs matériels. Cette étude, qui fait partie des perspectives de recherche à court terme, tirera certainement mieux partie de l'intérêt de la prise en compte de la longue mémoire par notre environnement.

8.5.2 Longue mémoire et performance des NoC

Cette section présente une étude expérimentale de l'impact de la longue mémoire sur les performances d'un réseau sur puce, centrée sur l'analyse du remplissage des files d'attente, bien que la section 8.5.1 conclut que la longue mémoire semble être peu importante dans le trafic des

processeurs. Cette est néanmoins intéressante car de la longue mémoire a été identifiée dans le trafic des accélérateurs matériels sur puce [151].

8.5.2.1 Simulations effectuées

Nous avons effectué ces expérimentations avec le réseau DSPIN (voir section 5.2.3.2), sur lequel nous avons connecté deux générateurs de trafic et une mémoire comme cela est resprésenté sur la figure 8.12(a). Le générateur "`LRD TG`" produit un trafic à longue mémoire, et le générateur "`Back TG`" est utilisé pour introduire une charge de fond sur le réseau. Nous nous intéressons ici uniquement à l'impact de la longue mémoire sur les performances du réseau, nous avons donc écrit manuellement les différents fichiers de configuration des générateurs de trafic, sans utiliser une trace de référence.

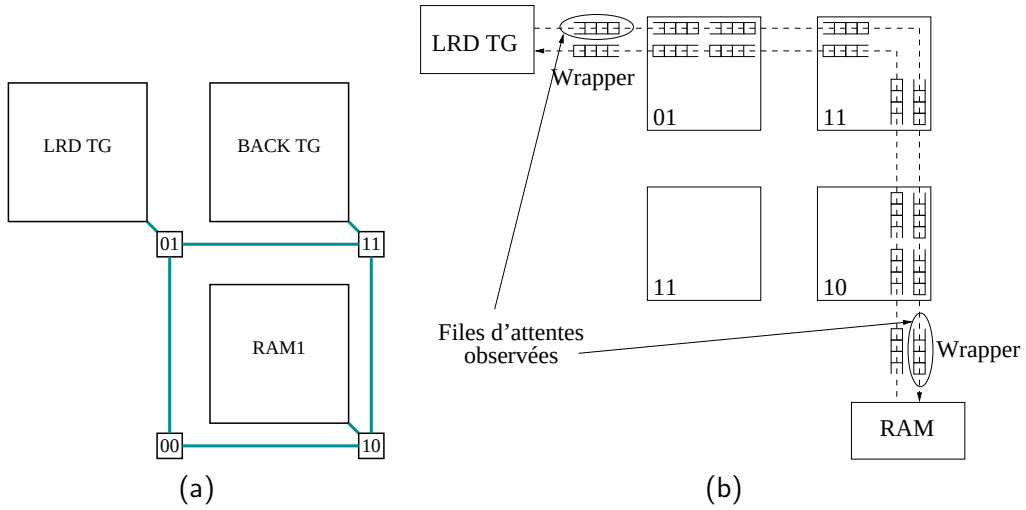


Fig. 8.12: Plateforme utilisée pour l'évaluation de l'impact de la longue mémoire sur les performance d'un NoC (a) et file d'attentes mises en jeu pour les communications du composant `LRD TG` (b).

La performance du réseau est observée par l'utilisation (taux de remplissage) des différentes files d'attentes présentes sur la plateforme comme cela est décrit sur la figure 8.12(b) pour les communications du composant `LRD TG`. On définit le taux de remplissage d'une file d'attente $T(k)$ comme la probabilité qu'il y ait k mots dans la file, pour k compris entre 0 et k_{max}. Pour calculer ce taux lors d'une simulation donnée, nous avons instrumenté la file d'attente, c'est à dire qu'à chaque cycle d'horloge, on enregistre combien de mots sont présents dans la file. A la fin de la simulation, on peut estimer la probabilité d'avoir k mots dans la file par sa fréquence d'apparition. Nous nous intéresserons au remplissage maximum (k_{max}) ainsi qu'au taux de remplissage moyen ($\mathbb{E}(T)$). D'autre part, afin d'avoir un autre indicateur de performance, nous présenterons aussi la latence (moyenne et écart-type) des communications, définie, comme cela est explicité dans la section 6.1, comme le nombre de cycle séparant le début de l'émission d'une requête, du début de la réception de la réponse associée.

Nous avons effectué deux types de simulations correspondant à deux mode de génération de trafic différents pour le composant `LRD TG` :

- **NO-SPLIT**. Ce mode est celui utilisé pour émuler le comportement d'un composant d'implémentant pas les transactions séparées (*split-transactions*). Dans ce cas le composant

attend systématiquement la réponse avant d'envoyer une nouvelle requête.

- **SPLIT**. Nous utilisons dans ce cas un mode de génération de trafic avec lectures et écritures non-blocantes, plus représentatif d'un composant flot de données (ASIC).

Par ailleurs, nous avons fixé une précision de niveau délai (plutôt qu'une précision niveau débit), toutes les requêtes sont à destination de l'unique mémoire présente dans la plateforme, et le générateur de trafic possède une seule phase. Les délais sont générés comme la réalisation d'un processus ayant la covariance d'un FGN de paramètre H variable et une loi marginale exponentielle de paramètre fixé à 5 cycles. La taille des transactions est générée comme la réalisation d'un processus IID exponentiel de paramètre 5 mots.

Nous avons utilisé différentes configurations du générateur de trafic `Back TG`, utilisé pour introduire une charge de fond constante variable sur le réseau. Nous avons effectué des simulations pour différentes valeurs de H. Toutes les simulations effectuées ont une durée de 10000 transactions du générateur de trafic `LRD TG`, ce qui est une durée assez longue pour permettre une bonne caractérisation du comportement des files d'attentes dans le réseau.

8.5.2.2 Résultats

Nous présentons ici les résultats sur la latence moyenne des transactions du composants `LRD TG`, ainsi que sur le remplissage moyen des deux files d'attentes indiquées sur la figure 8.12(b). Les autres files d'attente (celles des routeurs notamment), n'ont jamais été beaucoup utilisées car la charge de trafic circulant dans le réseau était trop faible. Ces deux files sont ci-après notées FA-TG (file d'attente du *wrapper* du générateur de trafic à longue mémoire) et FA-RAM (file d'attente du *wrapper* de la mémoire).

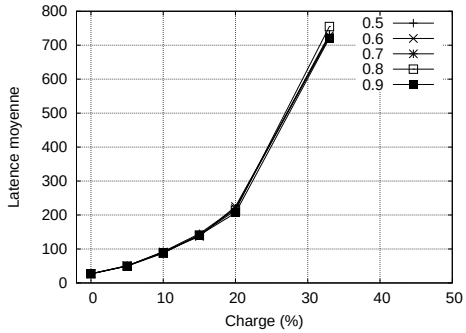


Fig. 8.13: Évolution de la latence moyenne en fonction de la charge induite par le générateur de trafic de fond pour différentes valeurs du paramètre de Hurst, dans le cas des simulations NO-SPLIT. Les différentes courbes sont complètement superposées.

Nous présentons dans un premier temps les résultats sur des simulations de type NO-SPLIT, pour lesquelles le générateur de trafic en émulant le comportement d'un cache (lecture blocante). Nous avons effectué ces simulations pour différentes valeurs du paramètre de longue mémoire H, ainsi que pour différentes charges induites par le générateur de trafic de fond. Le résultat est que le remplissage des files d'attentes ainsi que la latence du réseau n'est pas affectée pas la longue mémoire dans ce cas. En effet les courbes des figures 8.13 et 8.14 sont toutes quasiment superposées, indiquant que la valeur du paramètre de longue mémoire n'influence pas le remplissage des files d'attente FA-TG et FA-RAM. La décroissance du remplissage moyen de la file FA-TG s'explique par le fait que plus la charge de fond est importante, plus la latence du

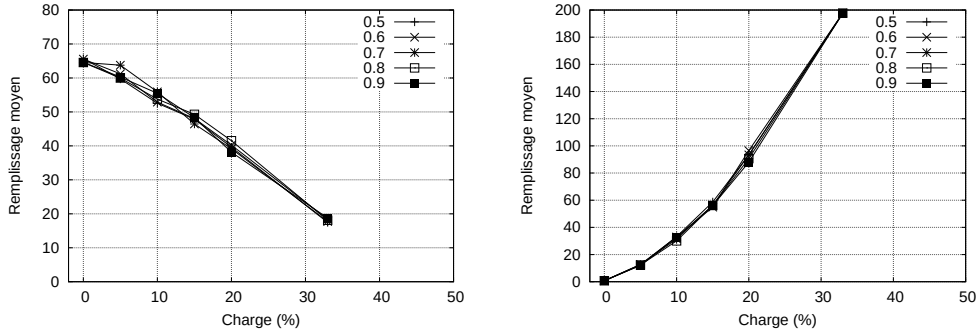


Fig. 8.14: Évolution du remplissage moyen de la file FA-TG (a) et de la file FA-RAM (b) en fonction de la charge induite par le générateur de trafic de fond dans le cas des simulations NO-SPLIT. Les différentes courbes sont superposées.

réseau est importante. Étant donné que le générateur de trafic attend la réponse avant d'émettre une nouvelle requête, il injecte plus lentement ses communications dans le réseau, le remplissage de la file du *wrapper* est donc décroissant. En revanche, il est normal que le taux de remplissage moyen de la file FA-RAM augmente, puisque les deux générateurs de trafic lui adressent leur communications. Il est à noter que la longue mémoire ne disparait pas du trafic, en effet, nous avons effectuer des analyses montrant que la longue mémoire est bien présente dans celui-ci.

On peut donc en conclure que la latence ainsi que le remplissage des files d'attentes ne varient pas avec le paramètre de longue mémoire. Ceci peut s'expliquer par le mode de communication utilisé (attente de la réponse avant de pouvoir émettre une nouvelle requête), qui entraîne un écart de temps important entre les différentes requêtes, dû en grande partie à la latence du réseaua. Le processus d'arrivée dans les files d'attente est dans ce cas trop faible en intensité pour que la caractéristique de longue mémoire puisse avoir un impact significatif. Ce résultat est important, il signifie que même si on identifie de la longue mémoire dans le trafic d'un composant fonctionnant dans le mode de communication NO-SPLIT (pas de transactions séparées), alors il n'est pas intéressant de la modéliser car son impact est complètement anéanti par la latence du réseau sur puce.

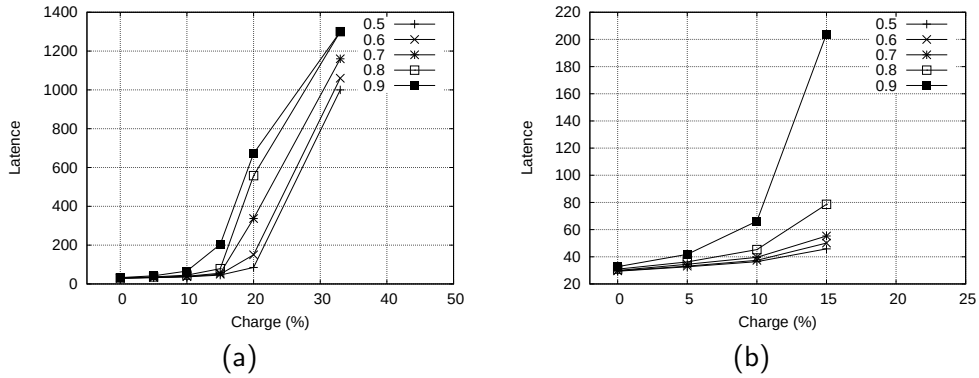


Fig. 8.15: Évolution de la latence moyenne en fonction de la charge induite par le générateur de trafic de fond dans le cas des simulations SPLIT, pour différentes valeurs de H. (b) est zoom du graphique (a).

Ce résultat n'est en revanche plus valable avec la deuxième configuration du générateur de trafic (SPLIT), qui autorise les transactions séparées (comportement typique d'un accélérateur matériel muni d'un accès direct à la mémoire). Dans ce cas, comme le montre les figures 8.15 et 8.16, la longue mémoire impacte de manière significative aussi bien la latence du réseau que le taux de remplissage des deux files d'attente considéres. En effet, le taux de remplissage moyen et maximum augmente avec le paramètre de Hurst, ce qui est en accord avec les résultats théoriques [118] et expérimentaux [42] existant à ce sujet.

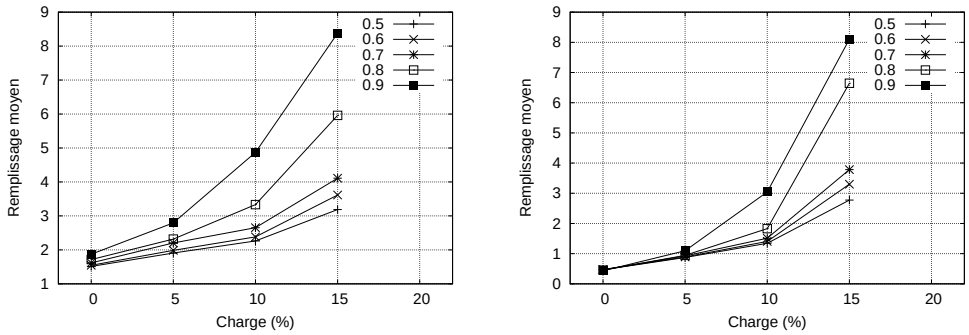


Fig. 8.16: Évolution du remplissage moyen de la file FA-TG (a) et de la file FA-RAM (b) en fonction de la charge induite par le générateur de trafic de fond dans le cas des simulations SPLIT, pour différentes valeurs de H.

On peut donc conclure deux observations expérimentales importantes quant à l'impact de la longue mémoire sur les performances d'un réseau sur puce :

- La longue mémoire dans le trafic émis par un composant n'autorisant pas les transactions séparées n'a pas d'impact sur les performances du réseau d'interconnexion que nous avons considéré. Il est dans ce cas inutile de la prendre en compte lors d'une modélisation de ce trafic qui a pour objectif le prototypage du réseau.
- La longue mémoire dans le trafic émis par un composant autorisant les transactions séparées a un impact significatif sur les performances du NoC que nous avons considéré. Il est dans ce cas indispensable d'en tenir compte au risque de sur-évaluer les performances du réseau.

Ces résultats vont prochainement faire l'objet d'une publication. Nous projetons aussi d'effectuer une étude expérimentale plus poussée de l'influence de la longue mémoire sur les performances des NoC, en diversifiant les réseaux sur puce d'une part, afin de s'assurer que ces résultats ne sont pas seulement valable pour l'interconnexion que nous avons choisit, et en diversifiant le type de trafic émis d'autre part en utilisant par exemple une génération de trafic au niveau débit.

8.6 Résumé des contributions expérimentales

Cette section propose un résumé des résultats expérimentaux qui ont été obtenus au cours de la thèse :

- La formalisation du trafic ainsi que les différents modes de communication intégrés à notre environnement de génération de trafic sur puce permettent, avec quelques restrictions, de

garantir la propriété qu'une trace collectée sur une interconnexion arbitraire peut être enregistrée et rejouée avec une autre interconnexion sans perte sensible de précision. Cette propriété permet donc une seule simulation de référence, à partir de laquelle différents générateurs peuvent être dérivés et utilisés avec des interconnexions diverses. Cette propriété est indispensable pour pouvoir utiliser notre environnement pour faire l'exploration architectural du réseau sur puce.

- L'algorithme de segmentation en phase proposé et décrit dans la section 6.2.3.2 effectue un découpage en phase pertinent et permet d'introduire le concept de génération de trafic stochastique multiphase.
- Les performances en terme de précision de cette méthode de génération se situent entre celles du rejeu (très bonne précision) et celle d'une charge aléatoire (très peu précise). Cette méthode offre donc un bon compromis et est une proposition sérieuse pour effectuer le prototypage des réseaux sur puce.
- La longue mémoire, bien qu'ayant été mise en évidence dans quelques morceaux de traces, n'est pas une caractéristique majeure du trafic des processeurs sur puce.
- L'impact de la longue mémoire sur les performances d'un réseau n'est significatif que dans le cas où elle est identifiée dans le trafic d'un composant utilisant les transactions séparées. Dans le cas contraire son effet est effacé à cause de la latence du réseau.

Conclusion et discussion

Nous avons dans cette partie exploré la problématique de la génération de trafic dans le cadre très spécifique des SoC. Cette problématique est directement liée à celle de la conception et du dimensionnement des interconnexions pour ces même SoC. Afin de pouvoir répondre à ces deux problèmes, nous avons mis au point et développé un environnement de génération de trafic sur puce permettant de faire l'exploration architecturale des futurs réseaux sur puce plus rapidement. Nous avons opté pour une approche basée sur l'analyse d'une trace de référence (simulation du composant que l'on souhaite remplacer par un générateur de trafic), cette analyse nous permettant de dériver semi-automatiquement un générateur de trafic adapté.

Lors de la mise en place de cet environnement, nous avons proposé un formalisme simple des communications ayant lieu au sein d'un SoC, qui est à la base de l'environnement. Nous avons aussi, en raison de la non-stationnarité du trafic émis par les processeurs, proposé une méthode de segmentation des traces de trafic en phases raisonnablement stationnaires. Cela nous a permis d'introduire le concept de génération de trafic stochastique multiphase, qui trouve sa place entre un rejeu d'une trace précédemment enregistrée, et la génération d'une charge aléatoire simple sur le réseau.

Dans la mise en place de cet environnement, nous nous sommes enfin attachés à garantir une propriété fondamentale, à savoir l'indépendance du trafic et de l'interconnexion. En effet, comme cet environnement est fait pour explorer les différents réseaux sur puce et les différents paramètres de ces réseaux, il est important que l'analyse de trafic (et donc le générateur qui en découle) soit indépendante de l'interconnexion. Nous avons clairement défini les restrictions à prendre en compte pour garantir cette propriété.

Nous avons enfin développé un générateur de trafic générique largement paramétrable dans SoCLib, qui nous a permis d'effectuer de nombreuses expérimentations qui constituent la contribution principale de cette partie. Le premier résultat est que le facteur d'accélération engendré par l'utilisation de générateurs de trafic n'est pas suffisant pour en justifier l'utilisation car la majeure partie du temps de simulation est passée dans l'interconnexion [SFR06b]. Concernant l'aspect multiphase, nous avons procédé à une exploration des différents paramètres qui nous a donné confiance dans l'automatisation de cette étape. L'algorithme de segmentation que nous avons proposé fournit des résultats satisfaisants, qui ont été mis en valeur par les performances de la génération de trafic stochastique multiphase [SFR06a], qui trouve ainsi sa place entre le rejeu et la génération d'une charge aléatoire simple. Nous avons enfin identifié dans certaines phases du trafic de la longue mémoire, mais sans que cela ne soit une caractéristique inévitable comme c'est le cas dans le trafic Internet. D'autre part nous avons montré avec des expérimentations simples, que la présence de longue mémoire dans le trafic des processeurs n'avait pas d'impact sur les performances du réseau, en raison de la latence de celui-ci et du mode de communication des processeurs. Ces résultats sont en cours de publication.

Cette partie comporte de nombreux développements logiciels sous licence GPL qui constituent une contribution importante de la thèse.

Les perspectives de recherches sont orientées vers une validation plus poussée de notre environnement de génération de trafic, en diversifiant les applications utilisées (algorithmes orientés contrôle, etc.), mais surtout en analysant le trafic d'autres types de composants (accélérateurs matériels entres autres). Nous projetons aussi d'évaluer notre méthode de segmentation sur d'autres traces de trafic, éventuellement de trafic Internet qui souffre aussi de non stationnarités. Une étude plus poussée de l'impact de la longue mémoire sur les performances des réseaux est aussi une perspective de recherche importante que nous comptons poursuivre. Enfin nous envisageons, afin de montrer l'intérêt global dans notre environnement, de réaliser complètement la phase de prototypage du réseau sur puce.

Conclusion générale

Cette thèse apporte des contributions dans deux problématiques de recherches différentes, mais complémentaires. La première est centrée sur l'analyse et la génération de trafic issu de réseaux de machine (comme Internet notamment), et la seconde concerne l'analyse et la génération du trafic émit par les composants intégrés au sein d'une même puce de silicium. Le dénominateur commun de ces deux problématiques est bien entendu l'analyse et la génération de trafic, et cette double expérience dans ce domaine m'a permis de proposer, aussi bien pour la génération de trafic Internet que pour la génération de trafic sur puce, des contributions originales.

Nous avons proposé, dans la première partie de ce manuscrit, une méthode de génération de réalisations de processus stochastiques à longue mémoire non-gaussiens Nous avons aussi proposé un modèle pour le trafic Internet (GAMMA – FARIMA), tenant compte de sa propriété de longue mémoire *et* de son aspect non-gaussien, ces deux caractéristiques lui étant unanimement reconnu dans la littérature. Nous avons montré que ce modèle est bien adapté au trafic agrégé, et ce pour une large gamme de niveau d'agrégation. Fort de la méthode de génération pré-citée, nous disposons aussi de méthodes pour produire du trafic synthétique, ce qui est un point important pour pouvoir générer du trafic en simulation et effectuer des évaluations de performance de réseaux. Étant donné que le modèle proposé reste valable pour différents niveaux d'agrégation, nous avons construit un modèle multirésolution du trafic, et nous avons montré que ce modèle permet de différencier un trafic ordinaire d'un trafic contenant une attaque de déni de service distribuée. Cela nous a permis de proposer une méthode de détection d'anomalie robuste car elle est insensible aux variations légitimes de trafic de par l'aspect multirésolution du modèle employé. Cette méthode à été mise à l'épreuve par des simulation réseaux, et cela nous a permis d'en cerner les limites concernant la détection en ligne d'anomalie. En effet, les résultats tendent à montrer qu'il faut pouvoir avoir accès un trafic agrégé très faiblement, et l'enregistrer sur une durée assez importante. Les perspectives de recherches inclus donc un nouveau travail sur une détection d'anomalie plus aisément intégrable à un équipement de supervision de réseaux. Nous avons enfin étudié expérimentalement le problème de la production de trafic à longue mémoire à partir d'un ensemble de sources. Une étude théorique a en effet montré que la superposition d'une infinité de sources émettant, pendant un temps infini, un trafic de type ON/OFF avec des temps ON distribués selon une loi à queue lourde, entraîne dans le trafic résultant de la longue mémoire dont le paramètre est très simplement relié à celui de la queue de la distribution. En pratique, nous avons montré que ce n'est pas tant le problème du nombre de sources que celui du temps de simulation qui empêche de retrouver expérimentalement la relation. Nous en avons donc déduit les bons paramètres de simulations à utiliser pour simuler un trafic à longue mémoire.

Dans la deuxième partie de la thèse, nous avons principalement proposé et mis en place un flot complet de génération de trafic sur puce dans le but d'effectuer le prototypage rapide du composant en charge de l'interconnexion, qui est en train de devenir un véritable réseau sur puce. Partant d'une trace de référence provenant d'une simulation d'un composant que l'on sou-

haite remplacer par un générateur de trafic, notre flot permet de dériver semi-automatiquement différents générateurs de trafic, en fonction des besoins du concepteur. Ce flot est basé sur une formalisation simple des communications ayant lieu au sein d'un système sur puce et afin de fournir un outil flexible et complet, nous avons défini différents modes de communications adaptés aux différents composants couramment intégrés dans les systèmes sur puce, ainsi que différents mode de génération de trafic (rejeu, stochastique, multiphase). Afin de valider notre approche, nous avons implémenté ce flot dans l'environnement de simulation académique SoCLib, en ne considérant que le trafic des processeurs. Cela nous a permis de faire de nombreuses expérimentations et d'obtenir différents résultats. Notre générateur de trafic peut tout d'abord rejouer une trace de référence enregistrée lors une simulation du composant. Le point important de ce rejeu est qu'il est, de par la formalisation choisie et en considérant certaines restrictions sur le mode de communication du composant, indépendant du réseau d'interconnexion. Ceci permet d'utiliser une trace collectée sur une interconnexion arbitraire et de la rejouer et permet surtout de faire le prototypage de l'interconnexion, ce qui correspond justement à en changer les paramètres. D'autre part, le trafic des processeurs sur puce n'est pas stationnaire, on voit au contraire apparaître des phases, correspondant à l'activité variable du processeur, et surtout à la mémoire cache qui lui est associée. Nous avons, à partir de cette observation, proposé un algorithme de segmentation de la trace de trafic en phases considérées comme stationnaires. Cela nous a permis d'introduire le concept de génération de trafic stochastique multiphase sur puce, qui trouve logiquement sa place entre le rejeu et la génération d'une charge aléatoire simple. Nous avons enfin analysé les traces de débit agrégé en vue d'identifier une propriété de longue mémoire. Notre résultat est que la longue mémoire, malgré qu'elle ait été identifiée sur certaines portions de traces, n'est pas une caractéristique importante du trafic des processeurs sur puce. Ce résultat a été complété par une étude expérimentale de l'impact de la longue mémoire sur la performance des réseaux sur puce, dont une des conclusions est que le mode de communication des processeurs empêche la longue mémoire d'avoir un impact sur le réseau. Les perspectives de recherches inclus l'analyse et la génération de trafic produit par des accélérateurs matériels, cela nous permettra de valider notre flot sur un autre type de composant. Aussi il y a de forte chances pour que de la longue mémoire soit présente pour les trafic de certains accélérateurs comme cela a été mis en évidence dans la littérature, nous pourrons alors réellement tirer profit du fait que notre environnement permet l'analyse et la génération de trafic à longue mémoire. A plus long terme, il est envisagé de relier l'analyse du trafic avec l'analyse de la consommation. Les communications étant une part importante de la consommation d'une puce, les caractéristiques statistiques du trafic ont de grandes chances de se retrouver dans l'évolution de la consommation.

Les deux parties de cette thèse comportent une analyse de grandes quantités de données, et c'est dans ce domaine que nous avons le plus appris et contribué au cours de cette thèse. La principale différence entre les données de la première et de la deuxième partie est que le trafic Internet provient d'un enregistrement fait par une tierce personne, alors que le trafic des processeurs sur puce a été crée par nos soins. Cette différence implique que le regard que l'on porte sur les données n'est pas le même dans le deux cas. Il est, de notre point de vue, plus délicat d'analyser des données que l'on crée soi-même, car les conclusions que l'on tire sont facilement biaisées par la connaissance accrue que l'on a du procédé même dont les données sont issues. Ce type de biais a été à l'origine de quelques difficultés au cours de la thèse, en particulier concernant l'analyse de trafic sur puce, il a fallu souvent trouver le bon compromis entre une analyse aveugle, ou l'on feint de ne rien connaître des données, et une analyse dépendante du fonctionnement des composants que nous avons simulés. Ce compromis est inhérent à tout travail expérimental, et nous nous sommes attachés à le faire de manière honnête, en particulier en produisant des expérimentations qui soient reproductible.

Bibliographie

[1] P. Abry, P. Flandrin, M.S. Taqqu, et D. Veitch. Wavelets for the analysis, estimation and synthesis of scaling data. Dans K. Park et W. Willinger, éditeurs, *Self-Similar Network Traffic and Performance Evaluation*. John Wiley & Sons, 2000.

[2] P. Abry et D. Veitch. Wavelet analysis of long-range dependent traffic. *IEEE Trans. on Info. Theory*, 44(1) :2–15, Janvier 1998.

[3] P.S. Addison. *Fractals and chaos : an illustrated course*. Institue of physics, 1997.

[4] Luca Carloni Alessandro Pinto et Alberto Sangiovanni-Vincentelli. Constraint-driven communication synthesis. Dans *Design Automation Conference (DAC)*, June 2002.

[5] VSI Alliance. Virtual component interface standard (ocb 2 1.0). En ligne : http://www.vsi.org/library/specs/summary.html, Avril 2001.

[6] A. Andersen et B. Nielsen. A markovian approach for modelling packet traffic with long range dependence. *IEEE journal on Selected Areas in Communications*, 5(16) :719–732, 1998.

[7] J. Archibald et J.-L. Baer. An economical solution to the cache coherence problem. Dans *11th International Symposium on Computer Architecture*, pages 355–362, Juin 1984.

[8] J. Archibald et J.-L. Baer. Cache coherence protocols : Evaluation using a multiprocessor simulation model. *ACM Transactions on Computer Systems*, 4 :273–298, Novembre 1996.

[9] W. J. Bainbridge. *Asynchronous System-on-Chip Interconnect*. Thèse de doctorat, University of Manchester, 2000.

[10] W. J. Bainbridge et S. B. Furber. Chain : A delay insensitive chip area interconnect. *IEEE Micro special issue on Design and Test of System on Chip*, 142, No.4. :16–23, Septembre 2002.

[11] C. Barakat, P. Thiran, G. Iannaccone, C. Diot, et P. Owezarski. A flow-based model for internet backbone traffic. Dans *ACM/SIGCOMM Internet Measurement Workshop*, pages 35–47, New York, NY, USA, 2002.

[12] Chadi Barakat, Patrick Thiran, Gianluca Iannaconne, Christophe Diot, et Philippe Owerzazki. Modeling Internet backbone traffic at the flow level. *IEEE Trans. on Signal Processing*, 2003.

[13] J.M. Bardet, G. Lang, G. Oppenheim, A. Philippe, et M.S. Taqqu. *Long-Range Dependence : Theory and Applications*, chapitre Generators of long-range dependent processes : A survey, pages 579–623. Birkhäuser, 2003.

[14] P. Barford, J. Kline, D. Plonka, et A. Ron. A signal analysis of network traffic anomalies. Dans *ACM/-SIGCOMM Internet Measurement Workshop*, Marseille, France, Novembre 2002.

[15] M. Basseville. Distance measures for signal processing and pattern recognition. *Signal Processing*, 18 :349–369, 1989.

[16] Luca Benini et Giovanni De Micheli. Powering networks on chips. Dans *ISSS*, pages 33–38, 2001.

[17] Luca Benini et Giovanni De Micheli. Analysis of power consumption on switch fabrics in network routers. Dans *Design Automation Conference (DAC)*, Mars 2002.

[18] Luca Benini et Giovanni De Micheli. Networks on chip : A new soc paradigm. *IEEE Computer*, 2002.

[19] J. Beran. *Statistics for Long-memory processes*. Chapman & Hall, New York, 1994.

[20] Davide Bertozzi et Luca Benini. Low power error resilient encoding for on-chip data buses. Dans *Design, Automation and Test Europe (DATE)*, 2002.

[21] Davide Bertozzi et Luca Benini. Xpipes : A Network-on-Chip Architecture for Gigascale Systems-on-Chip. *IEEE Circuits and Systems Magazine*, 4, 2004.

[22] Thomas Bohnert et Edmundo Monteiro. A comment on simulating lrd traffic with pareto on/off sources. Dans *CoNEXT'05 : Proceedings of the 2005 ACM conference on Emerging network experiment and technology*, pages 228–229, New York, NY, USA, 2005. ACM Press.

[23] M. Brière, L. Carrel, T. Michalke, F. Mieyeville, I. O'Connor, et F. Gaffiot. Design and behavioural modelling tools for optical networks on chip. Dans *Design, Automation and Test in Europe*, pages 738–749, Février 2004.

[24] M. Brière, F. Mieyeville, I. O'Connor, et F. Gaffiot. Un réseau d'interconnexion optique passif basé sur le routage en longueur d'onde. Dans *SympAAA*, France, Septembre 2003.

[25] P.J. Brockwell et R.A. Davis. *Time Series : Theory and Methods, 2nd edn.* Springer Series in Statistics, New York, 1991.

[26] J. Brutlag. Aberrant behavior detection in time series for network monitoring. Dans *USENIX System Administration Conference*, New Orleans, Décembre 2000.

[27] Kenneth P. Burnham et David Anderson. *Model Selection and Multi-Model Inference*. Springer ; 2 edition, 2003.

[28] J. Carroll et D. Long. *Theory of Finite Automata with an Introduction to Formal Languages*. Prentice Hall, Englewood Cliffs, 1989.

[29] L.M. Censier et P. Feautrier. A new solution to cache coherence problems in multicache systems. Dans *ieeetc*, pages 1112–1118, Décembre 1978.

[30] Henry Chang et Larry Cooke Grant Martin et al. *Surviving the SoC Revolution - A Guide to Platform-Based Design*. Kluwer Academic Publishers, Novembre 2000.

[31] C-M. Cheng, H.T. Kung, et K-S. Tan. Use of spectral analysis in defense against dos attacks. Dans *IEEE Globecom*, Taipei, Taiwan, 2002.

[32] ARM company Ldt. Amba specifications rev 2.0. En ligne : http://www.arm.com/armtech/AMBA, 1999.

[33] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, et A. Scandurra. Spidergon : a novel on-chip communication network. Dans *2004 International Symposium on System-on-Chip*, pages 15–22, Novembre 2004.

[34] M. Crouse et R. Baraniuk. Fast, exact synthesis of gaussian and nongaussian long-range-dependent processes. *IEEE Trans. on Info. Theory*, 1999. submitted.

[35] William J. Dally et Brian Towles. Route packets, not wires : On-chip interconnection networks. Dans *Design Automation Conference*, pages 684–689, 2001.

[36] Robert Davies. Newran c++ random number library. En ligne : http://www.robertnz.net/, Novembre 2005.

[37] N. Desaulniers-Soucy et A. Iuoras. Traffic modeling with universal multifractals. Dans *IEEE Globecom*, 1999.

[38] QoSMOS Traffic Designer. http ://www.qosmos.net.

[39] C. R. Dietrich et G. N. Newsam. Fast and exact simulation of stationary Gaussian processes through circulant embedding of the covariance matrix. *SIAM Journal on Scientific Computing*, 18(4) :1088–1107, 1997.

[40] P. Doukhan, G. Openheim, et M. Taqqu. *Theory and Applications of Long Range Dependance*. Birkhäuser, 2002.

[41] Allen B. Downey. The structural cause of file size distributions. Dans *MASCOTS '01 : Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01)*, page 361, Washington, DC, USA, 2001. IEEE Computer Society.

[42] A. Erramilli, O. Narayan, et W. Willinger. Experimental queueing analysis with long-range dependent packet traffic. *ACM/IEEE transactions on Networking*, 4(2) :209–223, 1996.

[43] Brad Calder et al. Simpoint. En ligne : http://www.cse.ucsd.edu/~calder/simpoint/, Avril 2001.

[44] M. Evans, N. Hastings, et B. Peacock. *Statistical Distributions*. John Wiley & Sons, Juin 2000.

[45] S. Farraposo, K. Boudaoud, L. Gallon, et P. Owezarski. Some issues raised by DoS attacks and the TCP/IP suite. Dans *SAR*, Batz-sur-mer, France, Juin 2005.

[46] Uriel Feige et Prabhakar Raghavan. Exact analysis of hot-potato routing. Dans *33rd Annual Symposium on Foundations of Computer Science*, pages 553–562. IEEE Computer Society Press, 1992.

[47] A. Feldmann, A. Gilbert, W. Willinger, et T. Kurtz. The changing nature of network traffic : Scaling phenomena. *ACM/SIGCOMM Computer Communication Review*, 2(28), Avril 1998.

[48] A. Feldmann, A.C. Gilbert, et W. Willinger. Data networks as cascades : Investigating the multifractal nature of internet wan traffic. Dans *ACM/SIGCOMM conference on Applications, technologies, architectures, and protocols for computer communication*, 1998.

[49] A. Feldmann et W. Whitt. Fitting mixtures of exponentials to long-tail distributions to analyze network performance models. *Performance Evaluation*, 31 :245–279, 1997.

[50] Stenio Fernandes, Carlos Kamienski, et Djamel Sadok. Accurate and fast replication on the generation of fractal network traffic using alternative probability models. *Performance and Control of Next-Generation Communications Networks*, 5244(1) :154–163, 2003.

[51] Wolfgang Fischer et Kathleen Meier-Hellstern. The markov-modulated poisson process (mmpp) cookbook. *Perform. Eval.*, 18(2) :149–171, 1993.

[52] N. Genko, D. Atienza, G. De Micheli, J. M. Mendias, R. Hermida, et F. Catthoor. A complete network-on-chip emulation framework. Dans *DATE 05*, pages 246–251, 2005.

[53] Wei-Bo Gong, Yong Liu, Vishal Misra, et Don Towsley. Self-similarity and long range dependence on the internet : a second look at the evidence, origins and implications. *Comput. Networks*, 48(3) :377–399, 2005.

[54] Alain Greiner et Pierre Guerrier. A generic architecture for on-chip paquets-switched interconnections. Dans *Design, Automation and Test in Europe*, 2000.

[55] Geoffrey R. Grimmett et David R. Stirzaker. *Probability and Random Processes, 3rd edition.* Oxford University Press, 2001.

[56] Independent JPEG Group. En ligne : http://ijg.org/, 2006.

[57] Pierre Guerrier. *Réseau d'Interconnexion pour Systèmes Intégrés.* Thèse de doctorat, Laboratoire d'Informatique de Paris VI, 2000.

[58] Greg Hamerly, Erez Perelman, et Brad Calder. How to use simpoint to pick simulation points. *Sigmetrics Perform. Eval. Rev.*, 31(4) :25–30, 2004.

[59] H. Michael Hauser et A. Wolfgang. The generation of stationary gaussian time series. Rapport technique, University of economics and business administration, Vienna, Austria, 1997.

[60] N. Hohn, D. Veitch, et P. Abry. Cluster processes, a natural language for network traffic. *IEEE Transactions on Signal Processing Special Issue on Signal Processing in Networking*, 8(51) :2229–2244, Octobre 2003.

[61] N. Hohn, D. Veitch, et P. Abry. Multifractality in tcp/ip traffic : the case against. *Computer Networks Journal*, to appear, 2005.

[62] Changcheng Huang, Michael Devetsikiotis, Ioannis Lambadaris, et A. Roger Kaye. Modeling and simulation of self-similar variable bit rate compressed video : a unified approach. Dans *Proceedings of SIGCOMM '95*, pages 114–125, New York, 1995. ACM Press.

[63] A. Hussain, J. Heidemann, et C. Papadopoulos. A framework for classifing denial of service attacks. Dans *SIGCOMM*, Karlsruhe, Germany, 2003.

[64] IBM. Coreconnect bus architecture. En ligne : http://www-3.ibm.com/chips/products/coreconnect/, 2001.

[65] Sonics Inc. En ligne : http://www.sonicsinc.com/, 2006.

[66] IPERF - The TCP/UDP Bandwidth Measurement Tool. En ligne : http://dast.nlanr.net/Projects/Iperf/.

[67] Raj Jain. *The Art of Computer Systems Performance Analysis.* John Wiley & Sons, 1991.

[68] Antoine Jalabert, Srinivasan Murali, Luca Benini, et Giovanni De Micheli. xpipesCompiler : A tool for instantiating application specific Networks on Chip. Dans *Design, Automation and Test in Europe*, Paris, France, Février 2004.

[69] Axel Jantsch et Hannu Tenhunen. *Networks on chip*. Kluwer, Février 2003.

[70] Rachid Jennane, Rachid Harba, et Gérard Jacquet. Méthodes d'analyse du mouvement brownien fractionnaire : théorie et résultats comparatifs. *Traitement du Signal*, 18(5-6) :419–436, 2001.

[71] S. Jin et D. Yeung. A covariance analysis model for ddos attack detection. Dans *IEEE International Conference on Communications*, Paris, France, Juin 2004.

[72] J2000 A jpeg2000 codec. En ligne : http://j2000.almacom.com/, 2006.

[73] J. Jung, B. Krishnamurthy, et M. Rabinovich. Flash Crowds and Denial of Service Attacks : Characterization and Implications for CDNs and Web Sites. Dans *International WWW Conference*, Honolulu, HI, Mai 2002.

[74] Gilles Kahn. The semantics of a simple language for parallel programming. Dans *IFIP Congress 74*, 1974.

[75] S. Kandula, D. Katabi, M. Jacob, et A. Berger. Botz-4-sale : survicing organized ddos attacks that mimic flash crowds. Dans A. Vahdat et D. Wetherall, éditeurs, *USENIX NSDI*, Boston, USA, Mai 2005.

[76] Faraydon Karim, Anh Nguyen, Sujit Dey, et Ramesh Rao. On-chip communication architecture for oc-768 network processors. Dans *Design, Automation and Test in Europe*, pages 678–683. ACM Press, 2001.

[77] R. H. Katz, S. J. Eggers, D. A. Wood, C. L. Perkins, et R. G. Sheldon. Implementing a cache consistency protocol. Dans *Proceedings of the 12th annual international symposium on Computer architecture*, pages 276–283. IEEE Computer Society Press, 1985.

[78] Arzad A. Kherani et Anurag Kumar. Long range dependence in network traffic and the closed loop behaviour of buffers under adaptive window control. *Perform. Eval.*, 61(2-3) :95–127, 2005.

[79] Donald Knuth. *The Art of Computer Programming, Volume 2 : Seminumerical Algorithms, Third Edition*, chapitre 3. Addison-Wesley, 1997.

[80] K. Lahiri, A. Raghunathan, et S. Dey. Evaluation of the traffic performance characteristics of system-on-chip communication architectures. Dans *Int. Conf. VLSI Design*, pages 29–35, 2001.

[81] Kanishka Lahiri, Anand Raghunathan, et Ganesh Lakshminarayana. LOTTERYBUS : A new high-performance communication architecture for system-on-chip designs. Dans *Design Automation Conference*, pages 15–20, 2001.

[82] A. Lakhina, M. Crovella, et C. Diot. Diagnosing network-wide traffic anomalies. Dans *SIGCOMM*, Août 2004.

[83] P. L'Ecuyer. Uniform random number generation. *Annals of Operations Research*, 53 :77–120, 1994.

[84] P. L'Ecuyer. Random number generation. Dans *Handbook of Simulation*, pages 93–137. John Wiley & Sons, 1998.

[85] Kangmin Lee, Se-Joong Lee, , et Hoi-Jun Yoo. A distributed on-chip crossbar switch scheduler for on-chip network. Dans *IEEE Custom Integrated Circuits Conference*, 2003.

[86] Se-Joong Lee, Seong-Jun Song, Kangmin Lee, Jeong-Ho Woo, Sung-Eun Kim, Byeong-Gyu Nam, et Hoi-Jun Yoo. An 800mhz star-connected on-chip network for application to systems on a chip. Dans *Proceedings International Solid State Circuits Conference (ISSCC)"*, 2003.

[87] J. Leijten et J van Meerbergen et al. Stream communication between real-time tasks in a high-performance multiprocessor. Dans *Design, Automation and Test in Europe*, Paris, France, Mars 1998.

[88] W. E. Leland, M. S. Taqqu, W. Willinger, et D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *ACM/IEEE transactions on Networking*, 2(1) :1–15, Février 1994.

[89] The Lex et Yacc Page. En ligne : http://dinosaur.compilertools.net/, 2006.

[90] L. Li et G. Lee. Ddos attack detection and wavelets. Dans *International Conference on computer communications and networks*, Août 2003.

[91] libmad MPEG audio decoder library. En ligne : http://www.underbit.com/products/mad/, 2006.

[92] Dake Liu, Daniel Wiklund, Erik Svensson, Olle Seger, , et Sumant Sathe. Socbus : The solution of high communication bandwidth on chip and short ttm. Dans *Real-Time and Embedded Computing Conference*, Gothenburg, Sweden, Septembre 2002.

[93] Mirko Loghi, Federico Angiolini, Davide Bertozzi, Luca Benini, et Roberto Zafalon. Analyzing on-chip communication in a mpsoc environment. Dans *DATE 04*, page 20752, 2004.

[94] S.B. Lowen, S.S. Cash, M. Poo, et M.C. Teich. Quantal neurotransmitter secretion rate exhibits fractal behavior. *The journal of Neuroscience*, 17(15) :5666–5677, Août 1997.

[95] m. stan et w. burleson. bus-invert coding for low-power i/o. Dans *ieee transactions on very large scale integration (vlsi) systems, 3 :49–58, 1995.*, volume 3, 1995.

[96] David J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. Available on-line at http://www.inference.phy.cam.ac.uk/mackay/itila/.

[97] J. MacQueen. Some methods for classification and analysis of multivariate observations. Dans *Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[98] Shankar Mahadevan, Federico Angiolini, Michael Storgaard, Rasmus Grøndahl Olsen, Jens Sparsø, et Jan Madsen. A network traffic generator model for fast network-on-chip simulation. Dans *DATE 05*, pages 780–785, 2005.

[99] B. B. Mandelbrot et M. S. Taqqu. Robust r/s analysis of long run serial correlation. Dans *42nd Session ISI*, page Book 2, New York, 1979.

[100] T. Marescaux, J-Y. Mignolet, A. Bartic, W. Moffat, D. Verkest, S. Vernalde, et R. Lauwereins. Networks on chip as hardware components of an os for reconfigurable systems. Dans *Field Programmable Logic and it's Applications*, Septembre 2003.

[101] T. Marescaux, S. Vernalde, A. Bartic, D. Verkest, S. Vernalde, et R. Lauwereins. Interconnection networks enable fine-grain dynamic multi-tasking on fpgas. Dans *Field Programmable Logic and it's Applications*, Septembre 2002.

[102] Gérard Mas et Philippe Martin. Network-on-chip : The intelligence is in the wire. Dans *22nd IEEE International Conference on Computer Design : VLSI in Computers & Processors (ICCD 2004)*, pages 174–177, 2004.

[103] MEDEA+. *SPIN : A scalable Network on Chip*, Novembre 2003.

[104] Benjamin Melamed. An overview of tes processes and modeling methodology. Dans *Performance/-SIGMETRICS Tutorials*, pages 359–393, 1993.

[105] Daniel Menard, Daniel Chillet, François Charot, et Olivier Sentieys. Automatic floating-point to fixed-point conversion for dsp code generation. Dans *International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES*, pages 270–276, 2002.

[106] M. Millberg, E. Nilsson, R. Thid, , et A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip. Dans *Design, Automation and Test in Europe*, Février 2004.

[107] M. Millberg, E. Nilsson, R. Thid, S. Kumar, , et A. Jantsch. The nostrum backbone - a communication protocol stack for networks on chip. Dans *VLSI Design Conference*, Janvier 2004.

[108] D. Moore, G.M. Voelker, et S. Savage. Inferring internet denial-of-service activity. Dans *Usenix Security Symposium*, 2001.

[109] L. Muscariello, M. Meillia, M. Meo, M.A. Marsan, et R.M. Cigno. An mmpp-based hierarchical model of internet traffic. Dans *IEEE International Conference on Communications*, pages 2143–2147, 2004.

[110] J. C. Nash. *Compact Numerical Methods for Computers : Linear Algebra and Function Minimisation, 2nd ed.*, chapitre 7, pages 84–93. Bristol, 1990.

[111] J.R. Norris. *Markov Chains*. Cambridge University Press, 1997.

[112] ns 2. The network simulator - ns-2. En ligne : http://www.isi.edu/nsnam/ns/, Juin 2006.

[113] OCP-IP. En ligne : http://www.ocpip.org/socket/ocpspec/, 2001.

[114] Computer Science Laboratory of Paris IV. Soclib simulation environment. En ligne : http://soclib.lip6.fr/, 2006.

[115] Open Microprocessor Initiative (OMI). Pi-bus draft standard specification (omi324), 1994.

[116] K. Park. On the effect and control of self-similar network traffic : a simulation perspective. Dans *Winter Simulation Conference*, pages 989–996, Décembre 1997.

[117] K. Park, G. Kim, et M. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. Dans *International Conference on Network Protocols*, page 171, Washington, DC, USA, 1996.

[118] Kihong Park et Walter Willinger, éditeurs. *Self-Similar Network Traffic and Performance Evaluation*. John Wiley & Sons, 2000.

[119] D.A. Patterson et J.L. Hennessy. *Organisation et conception des ordinateurs - L'interface matériel/logiciel*. DUNOD, 1994.

[120] S. Paulo, V. Rui, et P. António. Multiscale fitting procedure using markov modulated poisson processes. *Telecommunication Systems*, 23 (1/2) :123–148, Juin 2003.

[121] V. Paxon et S. Floyd. Wide-area traffic : The failure of poisson modeling. *ACM/IEEE transactions on Networking*, 3(3) :226–244, Juin 1995.

[122] V. Paxson. Bro : a system for detecting network intruders in real-time. *Computer Networks Journal*, 31(23–24) :2435–2463, 1999.

[123] Li-Shiuan Peh et William J. Dally. Flit-reservation flow control. Dans *6th International Symposium on High-Performance Computer Architecture*, pages 73–84, Janvier 2000.

[124] Dan Pelleg et Andrew Moore. X-means : Extending k-means with efficient estimation of the number of clusters. Dans *International Conference on Machine Learning*, pages 727–734, San Francisco, 2000.

[125] Santiago Gonzalez Pestana, Edwin Rijpkema, Andrei Rădulescu, Kees Goossens, et Om Prakash Gangwal. Cost-performance trade-offs in networks on chip : A simulation-based approach. Dans *DATE 04*, page 20764, 2004.

[126] Frédéric Pétrot et Pascal Gomez. Lightweight implementation of the posix threads api for an on-chip mips multiprocessor with vci interconnect. Dans *DATE 03 Embedded Software Forum*, pages 51–56, 2003.

[127] A. Pinto. Synthesis of on-chip networks. Mémoire de Master, UC Berkeley Electronics Research Lab, Juin 2003.

[128] A. Pinto, L.P. Carloni, , et A. Sangiovanni-Vincentelli. Iefficient synthesis of networks on-chip. Dans *International Conference on Computer Design*, 2003.

[129] R. Price. A useful theorem for non-linear devices having gaussian inputs. *IEEE Trans. Inform. Th.*, 4 :69–72, 1958.

[130] F. Pétrot et A. Greiner. Cache coherency and memory consistency in noc based shared memory multiprocessor soc architectures. Dans *4th International Seminar on Application Specific Multiprocessor SoC*, Saint Maximin la Sainte Baume, France, 2004.

[131] E. Rijpkema et K. G. W. Goossens. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. Dans *Design, Automation and Test in Europe*, Mars 2003.

[132] PM Robinson. Semiparametric analysis of long-memory time series. *Annals of Statistics*, 22 :515–539, 1994.

[133] Andrei Rădulescu et Kees Goossens. Communication services for networks on silicon. Dans *Domain-Specific Processors : Systems, Architectures, Modeling and Simulation*, pages 275–299. Marcel Dekker, dec 2003.

[134] rung-bin lin et chi-ming tsai. weight-based bus-invert coding for low-power applications. Dans *proc. of asp-dac/vlsi design 2002*, pages 121–125, Janvier 2002.

[135] Arteris SA. Arteris network on chip company. En ligne : http://www.arteris.net/, 2006.

[136] Paulo Salvador, Antonio Pacheco, et Rui Valadas. Modeling ip traffic : joint characterization of packet arrivals and packet sizes using bmaps. *Computer Networks*, 44(3) :335–352, 2004.

[137] S. Sarvotham, R. Riedi, et R. Baraniuk. Connection-level analysis and modeling of network traffic. Rapport technique, ECE Dept., Rice Univ., 2001.

[138] Timothy Sherwood, Erez Perelman, Greg Hamerly, Suleyman Sair, et Brad Calder. Discovering and exploiting program phases. *IEEE Micro*, 23(6) :84–93, 2003.

[139] David Sig&#252 ;enza-Tortosa, Tapani Ahonen, et Jari Nurmi. Issues in the development of a practical noc : the proteo concept. *Integr. VLSI J.*, 38(1) :95–105, 2004.

[140] Spice. The spice page. En ligne : http://www-asim.lip6.fr, 2006.

[141] Yi-Ran Sun. Simulation and performance evaluation for networks on chip. Mémoire de Master, Department of Microelectronics and Information Technology, Royal Institute of Technology, Décembre 2001.

[142] Yi-Ran Sun, Shashi Kumar, et Axel Jantsch. Simulation and evaluation for a network on chip architecture using ns-2. Dans *20th IEEE Norchip Conference*, Copenhagen, Novembre 2002.

[143] Inc. SuperH. Superhyway overview. En ligne : http://www.superh.com/products/shyway.htm, Juin 2003.

[144] M. Taqqu, V. Teverosky, et W. Willinger. Is network traffic self-similar or multifractal ? *Fractals*, 5(1) :63–73, 1997.

[145] R. Thid. A network on chip simulator. Mémoire de Master, Department of Microelectronics and Information Technology, Royal Institute of Technology, Août 2002.

[146] R. Thid, M. Millberg, et A. Jantsch. Evaluating NoC communication backbones with simulation. Dans *21th IEEE Norchip Conference*, Riga, Novembre 2003.

[147] Rikard Thid, KoIngo Sander, et Axel Jantsch. Flexible bus and noc performance analysis with configurable synthetic workloads. Dans *9th Euromicro Conference on Digital System Design (DSD 2006)*, Août 2006.

[148] TRINOO - Distributed network DoS tool. En ligne : http://staff.washington.edu/dittrich/misc/trinoo.analysis/.

[149] H.S. Vaccaro et G.E. Liepins. Detection of anomalous computer session activity. Dans *IEEE Symposium on Security and Privacy*, pages 280–289, Oakland, California, Mai 1989.

[150] Girish Varatkar et Radu Marculescu. Traffic analysis for on-chip networks design of multimedia applications. Dans *Design, Automation and Test in Europe*, pages 795–800, 2002.

[151] Girish Varatkar et Radu Marculescu. On-chip traffic modeling and synthesis for mpeg-2 video applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(1) :108–119, 2004.

[152] D. Veitch et P. Abry. A wavelet based joint estimator of the parameters of long-range dependence. *IEEE Trans. on Info. Theoryspecial issue on "Multiscale Statistical Signal Analysis and its Applications"*, 45(3) :878–897, Avril 1999.

[153] Daniel Wiklund. Implementation of a behavioral simulator for on-chip switched networks. Dans *Swedish System-on-Chip Conference (SSoCC)*, Falkenberg, Sweden, Mars 2002.

[154] Daniel Wiklund et Dake Liu. Switched interconnect for system-on-a-chip designs. Dans *IP2000 Europe conference*, Edinburgh, Scotland, Octobre 2000.

[155] Daniel Wiklund, Sumant Sathe, et Dake Liu. Network on chip simulations for benchmarking. Dans *IWSOC*, pages 269–274, 2004.

[156] W. Willinger, Murad S. Taqqu, R. Sherman, et Daniel V. Wilson. Self-similarity through high-variability : statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1) :71–86, 1997.

[157] Walter Willinger, Murad Taqqu, et Ashok Erramilli. A bibliographical guide to self-similar traffic and performance modeling for modern high-speed networks. *Stochastic Networks : Theory and Applications, Royal Statistical Society Lecture Notes Series*, 4, 1996.

[158] Wolfram. Gaussian integral. En ligne : http://mathworld.wolfram.com/GaussianIntegral.html, 2007.

[159] N. Ye. A markov chain model of temporal behavior for anomaly detection. Dans *Workshop on Information Assurance and Security*, West Point, NY, Juin 2000.

[160] Terry Tao Ye, Luca Benini, et Giovanni De Micheli. Packetized on-chip interconnect communication analysis for mpsoc. Dans *Design, Automation and Test in Europe*, pages 344–349, 2003.

[161] W.C. Yen, W.L. Yen, et K.-S. Fu. Data coherence problem in a multicache system. *IEEE Transactions on Computers*, 34(1), Janvier 1985.

[162] T. Yoshihara, S. Kasahara, et Y. Takahashi. Practical time-scale fitting of self-similar traffic with markov-modulated poisson process. *Telecommunication Systems*, 17 :185–211, 2001.

[163] J. Yuan et K. Mills. Ddos attack detection and wavelets. Rapport technique, National Institute of Standards and Technology, 2004.

Bibliographie Personnelle

Journaux internationaux

[SLB+07] Antoine Scherrer, Nicolas Larrieu, Pierre Borgnat, Philippe Owezarski and Patrice Abry. Non Gaussian and Long Memory Statistical Characterisations for Internet Traffic with Anomalies. Dans *IEEE Transactions on Dependable and Secure Computing (TDSC)*, Vol 4, No 1, pp. 56-70. Janvier 2007.

Conférences internationales avec comité de lecture

[SFR06a] Antoine Scherrer, Antoine Fraboulet, and Tanguy Risset. Automatic phase detection for stochastic on-chip traffic generation. Dans *CODES+ISSS*, pages 88–93, Séoul, Corée du sud, Octobre 2006.

[SFR06b] Antoine Scherrer, Antoine Fraboulet, and Tanguy Risset. Generic multi-phase on-chip traffic generator. Dans *ASAP*, pages 23–27, Steamboat Springs, USA, Septembre 2006.

[SLB+06a] Antoine Scherrer, Nicolas Larrieu, Pierre Borgnat, Philippe Owezarski, and Patrice Abry. Non gaussian and long memory statistical modeling of internet traffic. Dans *IPS MOME*, Salzbourg, Autriche, Mars 2006.

[FRS04] Antoine Fraboulet, Tanguy Risset, and Antoine Scherrer. Hardware-software fast and accurate prototyping with Soclib & MMAlpha. Dans Andy D. Pimentel and Stamatis Vassiliadis, editors, *Computer Systems : Architecture, Modeling, and Simulation (SAMOS 2004)*, volume 3133 of *Lecture Notes in Computer Science*, pages 453–462. Springer Verlag, Juillet 2004.

Conférences francophones avec actes

[BLO+06] P. Borgnat, N. Larrieu, P. Owezarski, P. Abry, J. Aussibal, L. Gallon, G. Dewaele, N. Nobelis, L. Bernaille, A. Scherrer, Y. Zhang, Y. Labit, and et al. Détection d'attaques de dénis de service par un modèle non gaussien multirésolution. Dans *CFIP*, Tozeur, Tunisie, Octobre 2006.

[SLB+06b] Antoine Scherrer, Nicolas Larrieu, Pierre Borgnat, Philippe Owezarski, and Patrice Abry. Une caractérisation non gaussienne et à longue mémoire du trafic internet et de ses anomalies. Dans *SAR*, Seignosse, France, Juin 2006.

[SA05] Antoine Scherrer and Patrice Abry. Marginales non gaussiennes et longue mémoire : analyse et synthèse de trafic internet. Dans *GRETSI 2005*, pages 279-282, Louvain la Neuve, Belgique, Septembre 2005.

[SFR04] Antoine Scherrer, Antoine Fraboulet, and Tanguy Risset. Hardware wrapper classification and requirements for on-chip interconnects. Dans *Signaux, Circuits et Systèmes 2004*, Monastir, Tunisie, mars 2004.

[SF03] Antoine Scherrer and Antoine Fraboulet. Etude de la couche transport des réseaux sur puces. Dans *SympAAA'2003*, pages 329–336, Nice, France, Mars 2003.

Rapports de recherches

[SFR05] Antoine Scherrer, Antoine Fraboulet, and Tanguy Risset. Analysis and synthesis of cycle-accurate on-chip traffic with long-range-dependence. Rapport de recherche 2005-53, LIP - ENS Lyon, 2005.

[SFR06c] Antoine Scherrer, Antoine Fraboulet, and Tanguy Risset. Multi-phase on-chip traffic generation environment. Rapport de recherche 2006-22, LIP - ENS Lyon, 2006.

[SLO+05] Antoine Scherrer, Nicolas Larrieu, Philippe Owezarski, Pierre Borgnat, and Patrice Abry. Non gaussian and long memory statistical characterisations for internet traffic with anomalies. Rapport de recherche 2005-35, LIP - ENS Lyon, 2005.

Annexe A
Calculs pour la synthèse de processus à longue mémoire non-gaussiens

Cette annexe présente le détails des calculs permettant de résoudre le problème posé à la section 3.1 : la synthèse de réalisations d'un processus Y ayant une covariance et une loi marginale prescrites. Nous supposons que l'on sait générer des réalisations de processus gaussiens ayant une covariance donnée. Les méthodes pour obtenir de tels réalisations sont décrites dans la section 2.8, page 40.

L'approche que nous avons suivie est de partir d'une famille de processus $\{X_i\}_{i=1,..,M}$ gaussiens indépendants (de moyenne nulle et de variance $\sigma_X^2 = 1$) ayant pour covariance γ_X, puis de prendre une fonction f de ces processus afin d'obtenir un processus Y ayant les caractéristiques que l'on souhaite, c'est à dire une loi marginale donnée *et* une covariance donnée γ_Y. Le problème consiste à trouver une relation entre γ_X et γ_Y.

On pourra alors générer M réalisations indépendantes de processus gaussiens ayant une covariance γ_X calculée en fonction de la covariance γ_Y, appliquer la fonction f et obtenir une réalisation du processus Y. Le calcul pour la loi du χ^2 de paramètre m est développé dans la section 3.1.2.1 page 49.

Cette annexe est organisée comme suit : la section A.1 présente les notations mathématiques utilisées, les sections A.2, A.3, A.4, A.5 et A.6 développent respectivement les calculs pour les lois exponentielle, Gamma, lognormale, uniforme et Pareto et enfin la section A.7 présente le théorème de Price ainsi que son utilisation pour ces calculs.

A.1 Notations

Dans toute cette annexe, les notations suivantes seront utilisées :

- $\forall i, \quad \{X_i[n]\}_{n\in\mathbb{N}}$ est un processus gaussien de moyenne nulle ($\mu_X = 0$) et de variance σ_X^2. Pour tout i, la covariance de ces processus sera toujours la même, notée γ_X :

$$\forall (i,j) \in \mathbb{N}^2, \forall k, \quad \gamma_{X_i}(k) = \gamma_{X_j}(k) \stackrel{\Delta}{=} \gamma_X(k)$$

- ρ_X désigne la fonction de corrélation des processus X_i, qui est reliée à la covariance par la relation : $\forall k, \rho_X(k) = \gamma_X(k)/\sigma_X^2$.
- $\{Y[n]\}_{n\in\mathbb{N}}$ est le processus dont on souhaite produire des réalisations synthétiques. Sa covariance sera notée γ_Y.

A.2 Loi exponentielle

On peut obtenir un processus Y suivant une loi exponentielle à partir de deux processus gaussiens indépendants en utilisant la relation suivante [44] :

$$\forall n, \quad Y[n] = \frac{\mu}{2}\left(X_1^2[n] + X_2^2[n]\right) \tag{A.1}$$

On peut montrer à partir de la définition d'une loi exponentielle que :

$$\forall n, \quad \mathbb{E}(Y[n]) = \mu$$

Par définition de la covariance du processus Y, on a :

$$\begin{aligned}
\gamma_Y(k) &= \mathbb{E}\big(Y[n]Y[n+k]\big) - \mathbb{E}\big(Y[n]\big)^2 \\
&= \frac{\mu^2}{4}\left[\mathbb{E}\Big(\big(X_1^2[n] + X_2^2[n]\big)\big(X_1^2[n+k] + X_2^2[n+k]\big)\Big)\right] - \mu^2 \\
&= \frac{\mu^2}{4}\left[\mathbb{E}\Big(2X_1^2[n]X_1^2[n+k] + 2X_1^2[n]X_2^2[n+k]\Big)\right] - \mu^2 \\
&= \frac{\mu^2}{4}\left[2 + 2\mathbb{E}\Big(X_1^2[n]X_1^2[n+k]\Big)\right] - \mu^2
\end{aligned} \tag{A.2}$$

On utilise ensuite la décomposition suivante [94] :

$$X_1[n+k] = \rho_X(k)X_1[n] + Z[n,k], \qquad \rho_X(k) = \frac{\gamma_X(k)}{\sigma_X^2} = \gamma_X(k)$$

Et on peut montrer que (voir section 3.1.2.1) :

- $\forall n, \quad Z[n,k]$ et $X_1[n]$ sont indépendantes.
- $\forall n, \quad Z[n,k]$ est gaussien de moyenne nulle.
- La variance de $Z[n,k]$ vaut $\sigma_X^2\Big(1-\rho_X^2(k)\Big)$.

En utilisant cette décomposition ainsi que le fait que $\mathbb{E}\big(X_1[n]^3\big) = 0$ et $\mathbb{E}\big(X_1[n]^4\big) = 3\sigma_X^2$ car $X_1[n]$ est gaussien, on obtient :

$$\begin{aligned}
\mathbb{E}\Big(X_1^2[n]X_1^2[n+k]\Big) &= \mathbb{E}\Big(X_1^2[n](\rho_X(k)X_1[n] + Z[n,k])^2\Big) \\
&= \rho_X^2(k)\mathbb{E}\big(X_1^4[n]\big) + \mathbb{E}\big(X_1^2[n]\big)\mathbb{E}\big(Z^2[n,k]\big) + \rho_X^2(k)\mathbb{E}\big(X_1^3[n]\big)\mathbb{E}\big(Z[n,k]\big) \\
&= 3\rho_X^2(k) + 1 - \rho_X^2(k) \\
&= 2\gamma_X^2(k) + 1
\end{aligned} \tag{A.3}$$

En reportant la relation (A.3) dans l'équation (A.2), on obtient finalement :

$$\boxed{\gamma_Y(k) = \mu^2\gamma_X^2(k)} \tag{A.4}$$

On peut inverser cette relation :

$$\boxed{\gamma_X(k) = \frac{\sqrt{\gamma_Y(k)}}{\mu}} \tag{A.5}$$

Ceci signifie que pour obtenir une loi exponentielle de paramètre μ, il va falloir synthétiser deux réalisations indépendantes de processus gaussiens de moyenne nulle, de variance unitaire et de covariance $\gamma_X(k)$ que l'on peut calculer en fonction de la covariance recherché en utilisant l'équation (A.5). En prenant la sommes des carrés de ces réalisations, on obtient la réalisation recherchée.

A.3 Loi gamma

Une loi gamma de paramètres α (paramètre de forme) et β (paramètre d'échelle) est définie, pour α entier, comme une somme de α variables aléatoires indépendantes suivant une loi exponentielle de paramètre β [44]. Soit $\{T_i\}_{i=1,..,M}$ une famille de processus indépendants exponentiels de paramètre $\mu = 1$, on a :

$$\forall n, \quad \sum_{i=1}^{\alpha} \beta T_i[n] \tag{A.6}$$

En utilisant l'équation (A.1), on a directement :

$$\forall n, \quad Y[n] = \sum_{i=1}^{\alpha} \frac{\beta}{2}\Big(X_{1,i}^2[n] + X_{2,i}^2[n]\Big) \tag{A.7}$$

On peut montrer que :

$$\forall n, \quad \mathbb{E}\big(Y[n]\big) = \alpha\beta$$

Les calculs sont très semblables à ceux correspondant à la loi du χ^2 développés dans la section 3.1.

$$\begin{aligned}
\gamma_Y(k) &= \mathbb{E}\big(Y[n]Y[n+k]\big) - \mathbb{E}\big(Y[n]\big)^2 \\
&= \mathbb{E}\Big(\sum_{i=1}^{\alpha} \beta T_i[n] \sum_{j=1}^{\alpha} \beta T_j[n+k]\Big) - \alpha^2\beta^2 \\
&= \beta^2 \sum_{i=1}^{\alpha}\sum_{j=1}^{\alpha} \mathbb{E}\Big(T_i[n]T_j[n+k]\Big) - \alpha^2\beta^2 \\
&= \beta^2 \sum_{i=1}^{\alpha} \Big[\mathbb{E}\big(T_i[n]T_i[n+k]\big) + \sum_{j=1,j\neq i}^{\alpha} \mathbb{E}\big(T_i[n]T_j[n+k]\big)\Big] - \alpha^2\beta^2 \\
&\quad \text{or} \quad T_i[n] \text{ et } T_j[n] \text{ sont indépendantes } \forall n, \forall (i,j), \quad i \neq j \\
\Rightarrow \gamma_Y(k) &= \beta^2 \sum_{i=1}^{\alpha} \Big[\mathbb{E}\big(T_i[n]T_i[n+k]\big) + \sum_{j=1,j\neq i}^{\alpha} \mathbb{E}\big(T_i[n]\big)\mathbb{E}\big(T_j[n+k]\big)\Big] - \alpha^2\beta^2
\end{aligned} \tag{A.8}$$

En utilisant les résultats de la section précédente sur la loi exponentielle (équation (A.4)), on peut montrer que :

$$\forall i, \quad \mathbb{E}\big(T_i[n]T_i[n+k]\big) = \gamma_X^2(k) + 1 \tag{A.9}$$

En reportant l'équation (A.9) dans l'équation (A.8), on obtient :

$$\gamma_Y(k) = \beta^2\Big(\alpha(\gamma_X^2(k)+1) + \alpha(\alpha-1)\Big) - \alpha^2\beta^2$$

$$\boxed{\gamma_Y(k) = \alpha\beta^2\gamma_X^2(k)} \tag{A.10}$$

On peut inverser cette relation :

$$\boxed{\gamma_X(k) = \sqrt{\frac{\gamma_Y(k)}{\alpha\beta^2}}} \tag{A.11}$$

Pour générer un processus suivant une loi gamma de paramètre α et β et ayant une covariance $\gamma_Y(k)$, il faudra donc générer 2α réalisations de processus gaussiens indépendantes de variance unitaire et de covariance γ_X calculée à partir de l'équation (A.11). Il suffira alors de faire la somme des carrés de tous ces processus afin d'obtenir le processus recherché.

A.4 Loi lognormale

Une loi lognormale correspond à l'exponentielle d'une une loi normale (gaussienne) [44]. Les paramètres de la loi lognormale sont la moyenne μ et la variance σ^2 de $X[n]$. On va donc dans ce cas considéré un processus gaussien noté X, de moyenne μ et de variance σ^2. On a donc par définition :

$$Y[n] = e^{X[n]} \tag{A.12}$$

On peut montrer que :

$$\forall n, \quad \mathbb{E}(Y[n]) = e^{\mu+\frac{\sigma^2}{2}} \stackrel{\Delta}{=} \mu_Y$$

Par définition de la covariance, on a :

$$\gamma_Y(k) = \mathbb{E}\big(Y[n]Y[n+k]\big) - \mathbb{E}\big(Y[n]\big)^2 \tag{A.13}$$

$$= \mathbb{E}\big(e^{X[n]}e^{X[n+k]}\big) - e^{2\mu+\sigma^2} \tag{A.14}$$

Nous allons utiliser la décomposition déjà introduite dans les sections précédentes, mais en tenant compte cette fois ci du fait que les $X[n]$ ont une moyenne non-nulle [94] :

$$X[n+k] = \rho_X(k)(X[n]-\mu) + \mu + Z[n,k], \qquad \rho_X(k) = \frac{\gamma_X(k)}{\sigma^2}$$

On peut montrer que (voir section 3.1.2.1) que :

- $\forall n, \quad Z[n,k]$ et $X[n]$ sont indépendantes.
- $\forall n, \forall k, \quad Z[n,k]$ est une variable aléatoire gaussienne de moyenne nulle.
- $\forall n, \forall k, \quad \mathbb{E}\big(Z[n,k]\big) = \sigma^2\Big(1-\rho_X^2(k)\Big)$.

En reportant ceci dans l'équation (A.14), on obtient :

$$\begin{aligned} &= \mathbb{E}\Big(e^{X[n](1+\rho_X(k))+Z[n,k]+\mu(1-\rho_X(k))}\Big) - \mathbb{E}\big(Y[n]\big)^2 \\ &= \mathbb{E}\Big(e^{X[n](1+\rho_X(k))}\Big) \times \mathbb{E}\Big(e^{Z[n,k]}\Big) \times e^{\mu(1-\rho_X(k))} - e^{2\mu+\sigma^2} \end{aligned} \tag{A.15}$$

Il faut maintenant calculer $\mathbb{E}\big(e^{Z[n,k]}\big)$ et $\mathbb{E}\big(e^{X[n]\left(1+\rho_X(k)\right)}\big)$. Pour cela nous utilisons quelques résultats de la théorie des probabilités. En particulier, on peut montrer que l'espérance d'une fonction d'une variable aléatoire X vaut :

$$\mathbb{E}\big(f(X)\big) = \int_{\mathbb{R}} f(x)g_X(x)dx, \quad g_X(x) \text{ étant la fonction de densité de X.}$$

On peut ainsi calculer $\mathbb{E}\big(e^{cX[n]}\big)$:

$$\begin{aligned}\mathbb{E}\big(e^{cX[n]}\big) &= \frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{+\infty} e^{cx}\exp\big(\frac{-(x-\mu_X)^2}{2\sigma^2}\big)dx \\ &= \frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{+\infty} \exp\big(cx-\frac{(x-\mu_X)^2}{2\sigma^2}\big)dx\end{aligned}$$

On obtient finalement :

$$\forall c\in\mathbb{R},\quad \mathbb{E}\big(e^{nX[n]}\big) = \exp\big(n\mu+\frac{n^2\sigma^2}{2}\big) \tag{A.16}$$

En utilisant la variance de $Z[n,k]$, on peut en déduire que :

$$\mathbb{E}\big(e^{Z[n,k]}\big) = \exp\big(\frac{\sigma^2}{2}\big(1-\rho_X^2(k)\big)\big) \tag{A.17}$$

$$\mathbb{E}\big(e^{X[n]\left(1+\rho_X(k)\right)}\big) = \exp\big(\frac{\sigma^2}{2}\big[1+\rho_X(k)\big]^2+\mu(1+\rho_X(k))\big) \tag{A.18}$$

En utilisant les équations (A.17) et (A.18) dans l'équation (A.15), on obtient :

$$\begin{aligned}\gamma_Y(k) &= \exp\Big[\frac{\sigma^2}{2}\big(1-\rho_X^2(k)\big)+\frac{\sigma^2}{2}\big(1+\rho_X(k)\big)^2+\mu\big(1+\rho_X(k)\big)+\mu\big(1-\rho_X(k)\big)\Big]-\exp\Big[2\mu+\sigma^2\Big] \\ &= \exp\Big[2\mu+\sigma^2\big(1+\rho_X(k)\big)\Big]-\exp\Big[2\mu+\sigma^2\Big] \\ &= e^{2\mu+\sigma^2}e^{\gamma_X(k)}-e^{2\mu+\sigma^2}\end{aligned}$$

$$\boxed{\gamma_Y(k) = \mu_Y^2\big(e^{\gamma_X(k)}-1\big)} \tag{A.19}$$

$$\boxed{\gamma_X(k) = \log\Big(1+\frac{\gamma_Y(k)}{\mu_Y^2}\Big)} \tag{A.20}$$

Afin de synthétiser un processus suivant une loi lognormale et une covariance donnée $\gamma_Y(k)$, il faut donc générer une réalisation d'un processus gaussien de moyenne μ et de variance σ^2 ayant une covariance que l'on peut calculer en fonction de γ_Y avec l'équation (A.20).

A.5 Loi Uniforme

Une loi uniforme de paramètre a,b peut être obtenue à partir de deux variables aléatoires gaussiennes indépendantes de moyenne nulle et de variance unitaire [44]. On peut ainsi montrer que pour obtenir un processus Y suivant une loi uniforme on doit prendre :

$$\forall n,\quad Y[n] = a+(b-a)\exp\Big(-\frac{1}{2}(X_1^2[n]+X_2^2[n]\Big) \tag{A.21}$$

On a par définition de la loi uniforme :

$$\forall n,\quad \mathbb{E}(Y[n]) = \frac{a+b}{2}$$

On procède alors de la même manière que pour une loi exponentielle

$$\begin{aligned}
\gamma_Y(k) &= \mathbb{E}\big(Y[n]Y[n+k]\big) - \mathbb{E}\big(Y[n]\big)^2 \\
&= \mathbb{E}\Big[\Big(a+(b-a)\exp\Big(-\frac{1}{2}(X_1^2[n]+X_2^2[n]\Big)\Big) \\
&\quad \times \Big(a+(b-a)\exp\Big(-\frac{1}{2}(X_1^2[n+k]+X_2^2[n+k]\Big)\Big)\Big] - \frac{(a+b)^2}{4} \\
&= a^2 + a(b-a)\Big[\mathbb{E}\Big(e^{-\frac{1}{2}(X_1^2[n]+X_2^2[n])}\Big) + \mathbb{E}\Big(e^{-\frac{1}{2}(X_1^2[n+k]+X_2^2[n+k])}\Big)\Big] \\
&\quad + (b-a)^2\exp\Big(-\frac{1}{2}\Big(X_1^2[n]+X_2^2[n]+X_1^2[n+k]+X_2^2[n+k]\Big)\Big) - \frac{(a+b)^2}{4}
\end{aligned} \tag{A.22}$$

Il faut maintenant calculer $\mathbb{E}\Big(e^{-\frac{1}{2}X^2[n]}\Big)$ et $\mathbb{E}\Big(e^{-\frac{1}{2}(X^2[n]+X^2[n+k])}\Big)$, $X[n]$ désignant une variable aléatoire gaussienne de moyenne nulle et de variance σ^2.

$$\begin{aligned}
\mathbb{E}\big(e^{-cX^2[n]}\big) &= \frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{+\infty} e^{-cx^2} e^{\frac{-x^2}{2\sigma^2}}\,dx \\
&= \frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{+\infty} e^{-x^2(c+\frac{1}{2\sigma^2})} \\
&= \frac{1}{\sqrt{2\pi\sigma^2}} \times (c+\frac{1}{2\sigma^2})^{-1/2}\sqrt{\pi} \\
\Rightarrow \forall c>0, \quad \mathbb{E}\big(e^{-cX^2[n]}\big) &= \frac{1}{\sqrt{2c\sigma^2+1}}
\end{aligned} \tag{A.23}$$

Pour calculer $\mathbb{E}\Big(e^{-\frac{1}{2}(X^2[n]+X^2[n+k])}\Big)$, nous allons utiliser la décomposition déjà introduite pour les lois précédentes :

$$X[n+k] = \rho_X(k)X[n] + Z[n,k], \qquad \rho_X(k) = \gamma_X(k)$$

On peut montrer que :

- $\forall n, \quad Z[n,k]$ et $X[n]$ sont indépendantes.
- $\forall n, \forall k, \quad Z[n,k]$ est une variable aléatoire gaussienne de moyenne nulle.
- $\forall n, \forall k, \quad \mathbb{E}\big(Z[n,k]\big) = \sigma^2\Big(1-\rho_X^2(k)\Big)$.

A partir de cette décomposition, on peut obtenir :

$$\begin{aligned}
\mathbb{E}\Big(e^{-\frac{1}{2}(X^2[n]+X^2[n+k])}\Big) &= \mathbb{E}\Big(\exp\big(-\frac{X^2[n]+\rho_X^2(k)X^2[n]+Z^2[n,k]+2\rho_X(k)X[n]Z[n,k]}{2}\big)\Big) \\
&= \mathbb{E}\Big(e^{\frac{-X^2[n]}{2}}\Big) \times \mathbb{E}\Big(e^{-\rho_X^2(k)X^2[n]}\Big) \times \mathbb{E}\Big(e^{\frac{-Z[n,k]^2}{2}}\Big) \times \mathbb{E}\Big(e^{-\rho_X(k)X[n]\mathbb{Z}[n,k]}\Big) \\
&= \frac{1}{\sqrt{4-\gamma_X^2(k)}}
\end{aligned} \tag{A.24}$$

En utilisant ce résultat dans l'équation (A.22), on obtient :

$$\begin{aligned}\gamma_Y(k) &= a^2 + a(b-a) + (b-a)^2\Big(\frac{1}{4-\gamma_X^2(k)}\Big) - \frac{(a+b)^2}{4} \\ &= (b-a)^2\Big(\frac{1}{4-\gamma_X^2(k)} + \frac{1}{4}\Big) \\ &= (b-a)^2\Big(\frac{1}{4-\gamma_X^2(k)} - \frac{1}{4}\Big)\end{aligned}$$

$$\boxed{\gamma_Y(k) = (b-a)^2\Big(\frac{\gamma_X^2(k)}{4(4-\gamma_X^2(k))}\Big)} \tag{A.25}$$

En inversant cette relation, on obtient finalement :

$$\boxed{\gamma_X(k) = 4\sqrt{\frac{\gamma_Y(k)}{(b-a)^2 + 4\gamma_Y(k)}}} \tag{A.26}$$

Afin de synthétiser un processus suivant une loi uniforme de paramètre a et b et une covariance donnée $\gamma_Y(k)$, il faut donc générer deux réalisations indépendante de processus gaussiens de moyenne nulle et de variance unitaire ayant une covariance que l'on peut calculer en fonction de γ_Y avec l'équation (A.26).

A.6 Loi Pareto

Une loi de Pareto de paramètre a, b peut être obtenue à partir de deux variables aléatoires gaussiennes indépendantes de moyenne nulle et de variance unitaire [44]. On peut ainsi montrer que pour obtenir un processus Y suivant une loi de Pareto on doit prendre :

$$\forall n, \quad Y[n] = b\exp\Big(\frac{X_1^2[n] + X_2^2[n]}{2a}\Big) \tag{A.27}$$

On peut montrer que :

$$\forall n, \quad \mathbb{E}(Y[n]) = \frac{ab}{a-1}$$

On précède comme pour les autres lois :

$$\begin{aligned}\gamma_Y(k) &= \mathbb{E}\big(Y[n]\,Y[n+k]\big) - \mathbb{E}\big(Y[n]\big)^2 \\ &= \mathbb{E}\Big(\Big(b^2\exp\Big(\frac{X_1^2[n] + X_2^2[n]}{2a}\Big)\Big) \\ &\quad \times \exp\Big(\frac{X_1^2[n+k] + X_2^2[n+k]}{2a}\Big)\Big) - \frac{(ab)^2}{(a-1)^2} \\ &= b^2\Big[\mathbb{E}\Big(\exp\Big(\frac{X_1^2[n] + X_1^2[n+k]}{2a}\Big)\Big)\Big]^2 - \frac{(ab)^2}{(a-1)^2}\end{aligned} \tag{A.28}$$

En utilisant la décomposition déjà introduite dans les sections précédentes :

$$X[n+k] = \rho_X(k)X[n] + Z[n,k], \qquad \rho_X(k) = \gamma_X(k)$$

On peut montrer en utilisant cette décomposition que :

$$\forall a > 2, \quad \mathbb{E}\Big(e^{\frac{X^2[n]+X^2[n+k]}{2a}}\Big) = \frac{a}{\sqrt{(a-1)^2 - \gamma_X^2(k)}} \tag{A.29}$$

En utilisant ce résultat dans l'équation (A.28), on obtient :

$$\boxed{\gamma_Y(k) = \frac{a^2 b^2 \gamma_X(k)}{(a-1)^4 - \gamma_X^2(k)(a-1)^2}} \tag{A.30}$$

En inversant cette relation, on obtient finalement :

$$\boxed{\gamma_X(k) = \frac{(a-1)^2\sqrt{\gamma_X(k)}}{a^2 b^2 + \gamma_X(k)(a-1)^2}} \tag{A.31}$$

Afin de synthétiser un processus suivant une loi de Pareto de paramètre $a > 2$ et b et une covariance donnée $\gamma_Y(k)$, il faut donc générer deux réalisations indépendante de processus gaussiens de moyenne nulle et de variance unitaire ayant une covariance que l'on peut calculer en fonction de γ_Y avec l'équation (A.31).

A.7 Utilisation du théorème de Price

Cette section présente le théorème de Price [129], développé pour l'étude des systèmes non-linéaire à entrées gaussiennes par Robert Price en 1958. On peut, en effet, utiliser ce résultat pour obtenir, d'une manière différentes, les résultats mathématique décrits précédemment dans cette annexe.

Soit Y, un processus de covariance γ_Y obtenue à partir d'un processus gaussien X de covariance γ_X tel que $Y = F(X)$, le théorème de Price permet d'exprimer les dérivées de γ_Y en fonction des dérivées de F :

$$\frac{\partial^n \gamma_Y(k)}{\partial \gamma_X(k)^n} = \mathbb{E}\Big[\frac{\partial^n F\big(X(m)\big)}{\partial X(m)^n}\frac{\partial^n F\big(X(m+k)\big)}{\partial X(m+k)^n}\Big] \tag{A.32}$$

En utilisant cette relation, on peut calculer la relation entre $\gamma_X(k)$ et $\gamma_Y(k)$ comme cela est illustré dans la section suivante.

A.7.1 Calculs pour la loi lognormale

Une loi lognormale correspond à l'exponentielle d'une une loi normale (gaussienne) [44]. Les paramètres de la loi lognormale sont la moyenne μ et la variance σ^2 de $X[n]$. On va donc dans ce cas considéré un processus gaussien noté X, de moyenne μ et de variance σ^2. On a donc par définition :

$$Y[n] = e^{X[n]} \tag{A.33}$$

On peut montrer que :

$$\forall n, \quad \mathbb{E}(Y[n]) = e^{\mu+\frac{\sigma^2}{2}} \triangleq \mu_Y$$

$$\forall n, \quad \mathbb{E}(Y^2[n]) - \mu_Y^2 = e^{2\sigma^2} - \mu_Y^2 \triangleq \sigma_Y^2$$

En appliquant le théorème de Price pour $n = 1$, on obtient :

$$\begin{aligned}\frac{\partial R(k)}{\partial \gamma_X(k)^n} &= \mathbb{E}\left(e^{X(m)} e^{X(m+k)}\right) \\ &= \mathbb{E}\left(e^{X(m)+X(m+k)}\right)\end{aligned}$$

A partir de résultats de la théorie des probabilités (espérance de l'exponentielle d'une variable aléatoire gaussienne), on peut montrer que :

$$\mathbb{E}\left(e^{X(m)+X(m+k)}\right) = e^{\sigma^2+\gamma_X(k)}$$

On a donc :

$$\gamma_Y(k) = e^{\sigma^2+\gamma_X(k)} + C$$

On peut alors calculer la valeur de C en prenant $k = 0$:

$$\begin{aligned}\gamma_Y(0) &= e^{\sigma^2+\gamma_X(0)} + C = \sigma_Y^2 \\ &= e^{2\sigma^2} + C = e^{2\sigma^2} - \mu_Y^2 \\ &\Rightarrow C = -\mu_Y^2\end{aligned}$$

On peut donc en déduire que :

$$\begin{aligned}\gamma_Y(k) &= e^{\sigma^2+\gamma_X(k)} - \mu_Y^2 \\ &= \mu_Y^2\left(e^{\gamma_X(k)} - 1\right)\end{aligned} \tag{A.34}$$

Ce résultat est bien le même que celui de la section A.4 (équation A.19).

A.7.2 Conclusions sur l'utilisation de ce théorème

L'utilisation de ce théorème peut, dans certain cas, simplifier les calculs. Cependant il ne permet pas de résoudre analytiquement le cas général ou la distribution de Y est arbitraire. D'autre part, le résultat n'est directement utilisable lorsque le processus Y est une fonction de plusieurs processus gaussiens. Cependant le fait qu'ils soient indépendants permet en général de séparer les termes de l'expression de $\gamma_Y(k)$ et d'appliquer le théorème de Price sur les différentes parties de l'équation.

Glossaire

AHB – *Advanced High-performance Bus, page 107*

APB – *Advanced Peripheral Bus, page 107*

ARIMA – *Integrated* ARMA, *page 35*

ARMA – *Auto-Regressive Moving-Average, page 18*

ASB – *Advanced System Bus, page 107*

ASIC – *Application Specific Integrated Circuit, page 98*

ASIP – *Application Specific Integrated Circuit, page 98*

BIC – *Bayesian Information Criterion page 26*

CABA – *Cycle Accurate, Bit Accurate, page 104*

CDF – *Cummulative Distribution Function, page 12*

CISC – *Complex Instruction Set Computer page 97*

DDoS – *Distributed Deny of Service, page 66*

DMA – *Direct Memory Access, page 98*

DoS – *Deny of Service, page 66*

DSP – *Digital Signal Processor, page 18*

DSPIN – *Distributed* SPIN *page 110*

FARIMA – *Fractionaly Integrated* ARMA, *page 18*

FBM – *Fractional Brownian Motion, page 32*

FGN – *Fractional Gaussian Noise, page 32*

FLIT – *Flot Transfer Unit page 108*

FPGA – *Field Programmable Gate Array page 120*

FSM – *Finite State Machine page 142*

FTP – *File Transfer Protocol, page 85*

GALS – *Glabally Asynchronous, Locally Synchronous page 97*

GCC – *GNU C Compiler page 147*

GPL – *GNU Public License page 51*

GPP – *General Purpose Processor, page 97*

HTTP – *HyperText Transfer Protocol page 69*

IDCT – *Inverse Discrete Cosine Transform page 121*

IDS – *Intrusion Detection System, page 66*

IID – Indépendants Indentiquement Distribués, *page 20*

IP – *Intellectual Proterty, page 102*

IR – *Instruction Register, page 97*

ISS – *Instruction Set Simulator, page 104*

LD – *Logscale Diagram, page 39*

LFU – *Least Frequently Used page 118*

LRD – *Long Range Dependence, page 30*

LRU – *Least Recently Used page 118*

MC – *Motion Compensation page 121*

METROSEC – Projet Métrologie et Sécurité pour l'Internet, *page 6*

MLE – *Maximum Likelihood Estimation, page 27*

MMPP – *Markov Modulated Poisson Process, page 22*

MPEG – *Moving Picture Expert Group, page 98*

MPSOC – *Multi-Processor* SOC, *page 103*

MPTG – *Multi-Phase Traffic Generator, page 139*

MQD – *Mean Quadratic Distance, page 75*

NOC – *Network-on-Chip, page 93*

NOSTRUM – Réseau sur puce développé en Suède, *page 110*

NS-2 – *Network Simulator*, version 2, *page 60*

OCP – *Open Core Protocol, page 103*

OS – *Operating System page 100*

PC – *Program Counter, page 97*

PDA – *Personal Digital Assistant, page 96*

PDF – *Probability Distribution Function, page 12*

QOS – Qualité de Service, *page 60*

RAM – *Random Access Memory page 99*

RENATER – Réseaux des universités françaises, *page 68*

RISC – *Reduced Instruction Set Computer page 97*

ROC – , *page 77*

RTL – *Register Transfer Level, page 104*

RTT – *Round Trip Time, page 67*

SMTP – *Simple Mail Transfer Protocol page 70*

SOC – *System-on-Chip, page 6*

SOCLIB – Bibliothèque *open-source* de composants, *page 7*

SPIN – *Scalable Programmable Interconnexion Network, page 110*

SYSTEMC – *SystemC, page 104*

TCP – *Transport Control Protocol, page 61*

TDMA – *Time Division Multiple Access, page 107*

TDN – *Temporally Disjoint Networks page 111*

TG – *Traffic Generator, page 127*

TLM – *Transaction Level Modelling, page 104*

TLM-T – *Timed Transaction Level Modelling, page 104*

UDP – *User Datagram Protocol, page 61*

USB – *Universal Serial Bus page 98*

VA – Variable Aléatoire, *page 12*

VC – *Virtual Component, page 102*

VCD – *Value Change Dump page 128*

VCI – *Virtual Component Interface, page 103*

Résumé

Cette thèse est composée de deux parties. La première explore la problématique de la modélisation de trafic Internet. Nous avons proposé, à partir de l'étude de nombreuses traces, un modèle basé sur des processus stochastiques non-gaussiens à longue mémoire (Gamma-Farima) permettant de modéliser de manière pertinente les traces de débit agrégé, et ce pour une large gamme de niveau d'agrégation. Afin de pouvoir générer du trafic synthétique, nous avons proposé une méthode de synthèse de tels processus. Nous avons ensuite, à partir du modèle Gamma-Farima, proposé un modèle multirésolution permettant de différencier un trafic régulier, d'un trafic contenant une attaque (de type déni de service distribuée). Ceci nous a permis de proposer une méthode de détection d'anomalie que nous avons évalué sur des traces réelles et en simulation. Enfin nous avons étudié expérimentalement le problème de la production de trafic à longue mémoire dans un simulateur de réseaux (NS-2). La deuxième partie traite la problématique de la génération de trafic au sein des systèmes sur puce (SoC). Dans ce domaine, l'arrivée de véritable réseaux sur puce place la conception de l'interconnexion au premier plan, et pour accélérer les simulations, il convient de remplacer les composants par des générateurs de trafic. Nous avons mis en place un environnement complet de génération de trafic sur puce permettant de rejouer une trace, de produire une charge aléatoire sur le réseau, de produire un trafic stochastique ajusté sur une trace de référence et de tenir compte des phases dans le trafic. Nous avons effectué de nombreuses simulations dans l'environnement de simulation de SoC académique SocLib qui nous ont permis de valider notre approche de la génération de trafic, d'évaluer notre algorithme de segmentation ainsi que la génération de trafic stochastique multiphase que nous avons introduite. Nous avons aussi exploré la présence de longue mémoire dans le trafic des processeurs sur puce, ainsi que l'impact de cette caractéristique sur les performances du réseau sur puce.

Abstract

This PhD is composed of two main parts. The first one focuses on Internet traffic modelling. From the analysis of many traffic traces, we have proposed a parsimonious model (Gamma-Farima) adapted to aggregated throughput traces and valid for wide range of aggregation levels. In order to produce synthetic traffic from this model, we have also studied the generation of sample path of non-gaussian and long memory stochastic processes. We have then used the Gamma-Farima model in order to build an anomaly detection method. To this end we have introduced a multiresolution model that can differentiate a regular traffic from a malicious one (including a DDoS attack). This method was evaluated both on real traces and simulations. Finally, we have studied the production of long range dependent traffic in a network simulator (NS-2). The second part of this PhD deals with the analysis and synthesis of on-chip traffic, *i.e.* the traffic occurring in a system on chip. In such systems, the introduction of networks on chip (NoC) has brought the interconnection system on top of the design flow. In order to prototype these NoC rapidly, fast simulations need to be done, and replacing the components by traffic generators is a good way to achieve this purpose. So, we have set up and developed a complete and flexible on-chip traffic generation environment that is able to replay a previously recorded trace, to generate a random load on the network, to produce a stochastic traffic fitted to a reference trace and to take into account traffic phases. Indeed most of the traffic traces we have obtained were non-stationary, we therefore need to split them into reasonably stationary parts in order to perform a meaningful stochastic fit. We have performed many experiments in the SocLib simulation environment that demonstrate that i) our traffic generation procedure is correct, ii) our segmentation algorithm provides promising results and iii) multiphase stochastic traffic generation is a good tradeoff between replay and simple random traffic generation. Finally, we have investigated the presence of long memory in the trace as well as the impact of long memory on the NoC performance.